THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

AReaL

On the Opportunities of Exploiting LLM Agents for ML System Design and Implementation

Binhang Yuan

Assistant Professor @ CSE, HKUST

Leader of AReaL Open-Source Community

05.09.2026

Brief Bio: Binhang Yuan

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

1 Ph.D.

Rice University — Computer Science

2013 – 2020

Advisor: Prof. Chris Jermaine

2 Postdoc

ETH Zürich — Computer Science

2021 – 2023

Advisor: Prof. Ce Zhang

3 Assistant Professor

HKUST — CSE

2023 – present

Relaxed System Lab · AReaL community

Research focus as Assistant Professor

The last 3 years (2023 – 2026)

System for LLM

AReaL · HexGen · HexiScale — building systems for LLMs

The next 3 years (2026 – 2029)

LLM for System

ARGUS · Autopoiesis · AReaL AutoPilot — agents engineering systems

The Path To a Foundation Model

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Data Preparation

Preparing a massive corpus for pretraining

A multi-stage pipeline:
- Acquisition (e.g. OCR)
- Cleaning;
- Filtering;
- De-duplication;
- Sampling;
- Shuffling.

Pre-training

Training a foundation model by fitting the large corpus by auto-regression

Core infra:
a large-scale parallel training system running over thousands of GPUs

Tuning & Alignment

Tune the model so it can follow user instructions effectively and safely.

Various methods:

- SFT
- RLHF
- RLAIF

Evaluation

Evaluate the model from different aspects:

- Comprehension
- Instruction following
- Expression
- Reasoning
- Coding
- Interaction
- Safety

Deployment

Deploy the model to serve users under different settings:
- Cloud
- Local devices

Require system efforts to optimize the performance, cost, and latency.

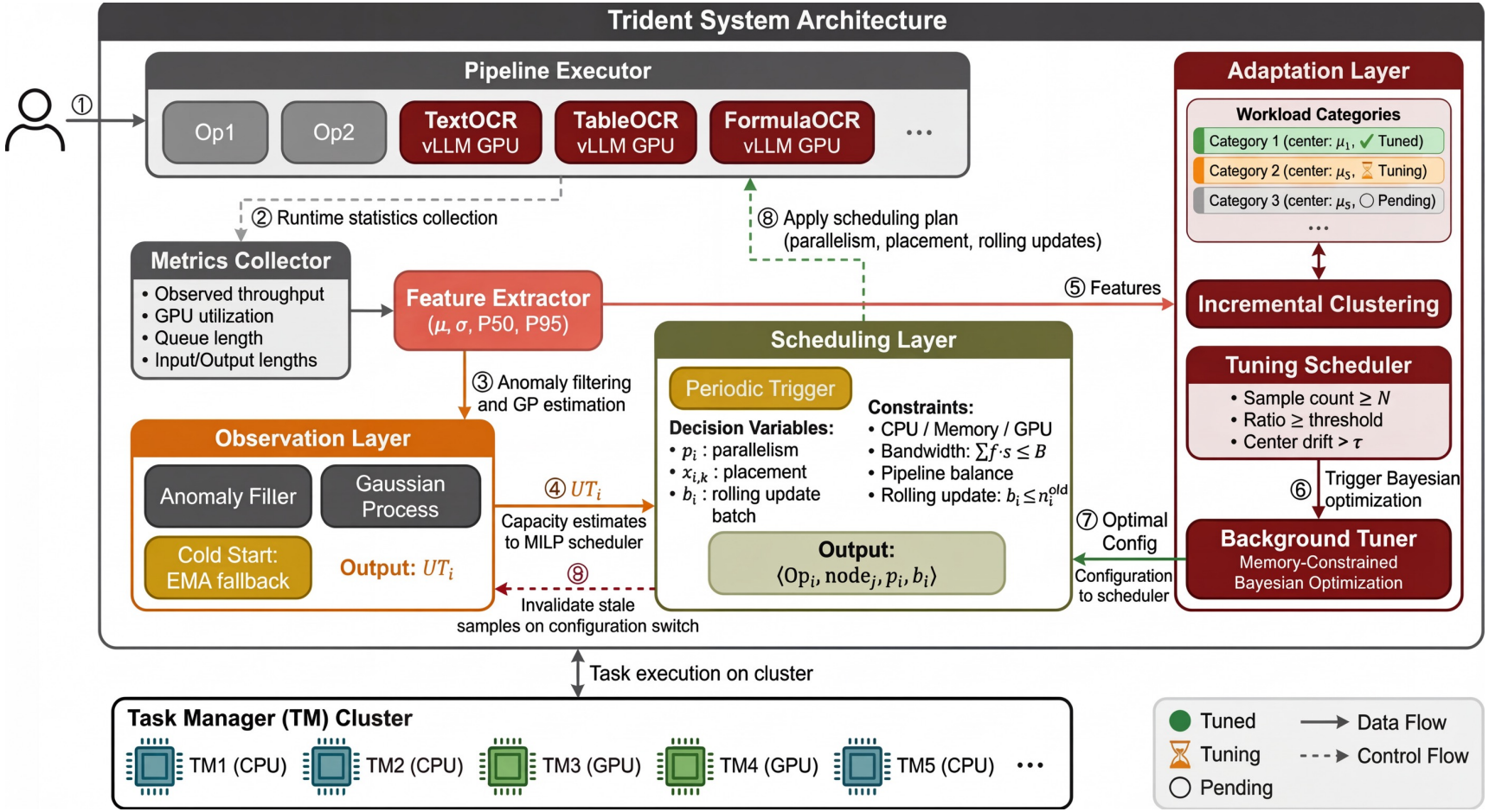
Representative works for at each stage:

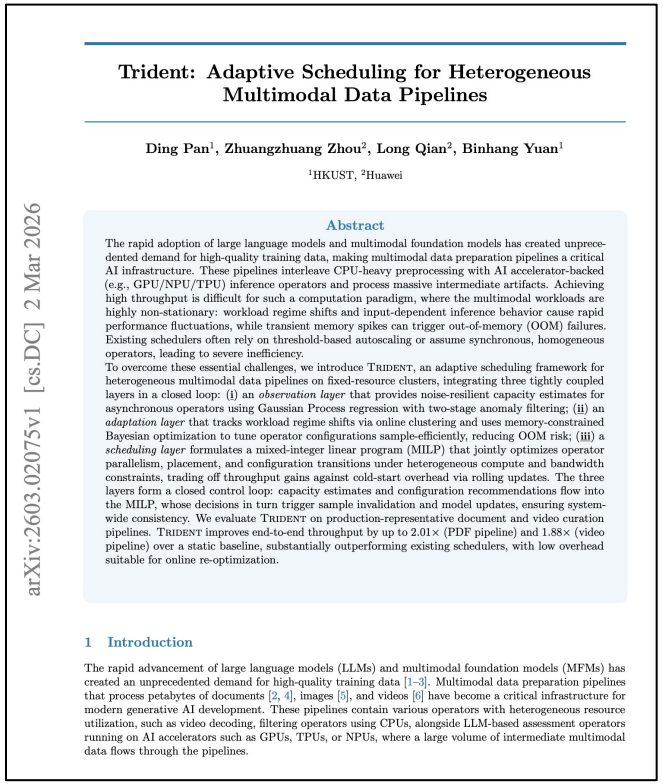
[Trident: VLDB 2026] [HexiScale: MLSys 2026] [AReaL: NeurIPS 2025, AReaL-DTA: ICML 2026] [AtmosSciBench: NeurIPS 2025] [HexGen: ICML 2024, HexGen-2: ICLR 2025, HexGen-3: ICML 2026]

Data Preparation System: Trident

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB





arXiv:2603.02075v1 [cs.DC] 2 Mar 2026

Trident: Adaptive Scheduling for Heterogeneous Multimodal Data Pipelines

Ding Pan[1], Zhuangzhuang Zhou[2], Long Qian[2], Binhang Yuan[1]

[1]HKUST, [2]Huawei

Abstract

The rapid adoption of large language models and multimodal foundation models has created unprecedented demand for high-quality training data, making multimodal data preparation pipelines a critical AI infrastructure. These pipelines interleave CPU-heavy preprocessing with AI accelerator-backed (e.g., GPU/NPU/TPU) inference operators and process massive intermediate artifacts. Achieving high throughput is difficult for such a computation paradigm, where the multimodal workloads are highly non-stationary: workload regime shifts and input-dependent inference behavior cause rapid performance fluctuations, while transient memory spikes can trigger out-of-memory (OOM) failures. Existing schedulers often rely on threshold-based autoscaling or assume synchronous, homogeneous operators, leading to severe inefficiency.

To overcome these essential challenges, we introduce TRIDENT, an adaptive scheduling framework for heterogeneous multimodal data pipelines on fixed-resource clusters, integrating three tightly coupled layers in a closed loop: (i) an *observation layer* that provides noise-resilient capacity estimates for asynchronous operators using Gaussian Process regression with two-stage anomaly filtering; (ii) an *adaptation layer* that tracks workload regime shifts via online clustering and uses memory-constrained Bayesian optimization to tune operator configurations sample-efficiently, reducing OOM risk; (iii) a *scheduling layer* formulates a mixed-integer linear program (MILP) that jointly optimizes operator parallelism, placement, and configuration transitions under heterogeneous compute and bandwidth constraints, trading off throughput gains against cold-start overhead via rolling updates. The three layers form a closed control loop: capacity estimates and configuration recommendations flow into the MILP, whose decisions in turn trigger sample invalidation and model updates, ensuring system-wide consistency. We evaluate TRIDENT on production-representative document and video curation pipelines. TRIDENT improves end-to-end throughput by up to 2.01× (PDF pipeline) and 1.88× (video pipeline) over a static baseline, substantially outperforming existing schedulers, with low overhead suitable for online re-optimization.

1 Introduction

The rapid advancement of large language models (LLMs) and multimodal foundation models (MFMs) has created an unprecedented demand for high-quality training data [1–3]. Multimodal data preparation pipelines that process petabytes of documents [2, 4], images [5], and videos [6] have become a critical infrastructure for modern generative AI development. These pipelines contain various operators with heterogeneous resource utilization, such as video decoding, filtering operators using CPUs, alongside LLM-based assessment operators running on AI accelerators such as GPUs, TPUs, or NPUs, where a large volume of intermediate multimodal data flows through the pipelines.

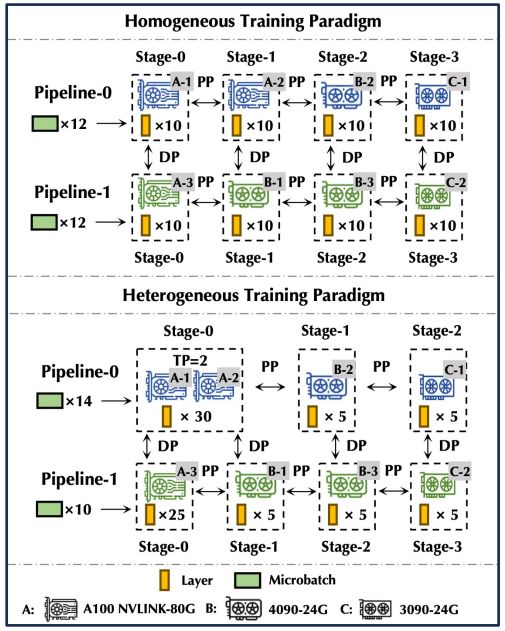
[VLDB 2026]

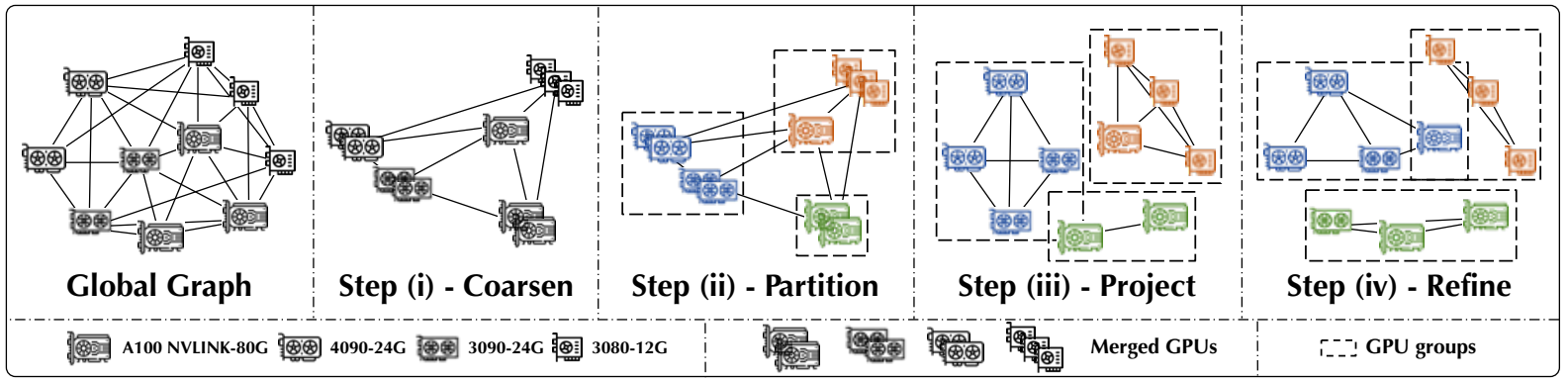
Large Scale Pretraining System: HexiScale

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

- A heterogeneous LLM training system that supports:
  - Data parallelism;
  - Pipeline parallelism;
  - Tensor model parallelism;
- A scheduling algorithm:
  - Two-phase graph partition algorithm.





HEXISCALE: FACILITATING LARGE LANGUAGE MODEL TRAINING OVER HETEROGENEOUS HARDWARE

Ran Yan [*1] Youhe Jiang [*1] Xiaonan Nie [2] Fangcheng Fu [3] Bin Cui [2] Binhang Yuan [1]

ABSTRACT

Training large language models (LLMs) is a computationally intensive task, which is typically conducted in data centers with homogeneous high-performance GPUs. In this paper, we explore an alternative approach by deploying training computations across heterogeneous GPUs to enable better flexibility and efficiency for heterogeneous resource utilization. Toward this end, we propose a novel system, HEXISCALE, that can flexibly support asymmetric partition of training computations in the scope of data-, pipeline-, and tensor model parallelism. We further formalize the allocation of asymmetric partitioned training computations over a set of heterogeneous GPUs as a constrained optimization problem and propose an efficient hierarchical graph partitioning algorithm. Our approach effectively allocates training computations across heterogeneous GPUs, fully leveraging the available computational power. We compare the performance of HEXISCALE with state-of-the-art homogeneous and heterogeneous training systems. When training LLMs at different scales (from 7B to 30B), empirical results demonstrate that: (i) compared to state-of-the-art homogeneous baselines running over homogeneous GPUs, HEXISCALE achieves *similar* performance when running over heterogeneous GPUs with the *same* theoretical FLOPS; (ii) compared to state-of-the-art heterogeneous baselines running on the same heterogeneous clusters, HEXISCALE delivers 1.5× to 2.4× higher throughput.

1 INTRODUCTION

Over the past few years, large language models (LLM) have demonstrated impressive performance and sparked a new wave of exciting AI applications (Bommasani et al., 2021). However, training these LLMs, such as GPT (OpenAI, 2024), Claude (Anthropic, 2024), Gemini (Reid et al., 2024), Llama (Touvron et al., 2023; Dubey et al., 2024), Mixtral (Jiang et al., 2024a), Yi (Young et al., 2024), Falcon (Institute, 2023) etc., can be extremely computation-intensive, often involving thousands of GPUs running for months. The high cost of deploying such training tasks in a cluster with homogeneous GPUs has become an obvious obstacle limiting the evolution of LLMs. We explore an alternative approach by *distributing training computations across heterogeneous GPUs*, enabling greater flexibility in resource utilization and democratizing LLM training.

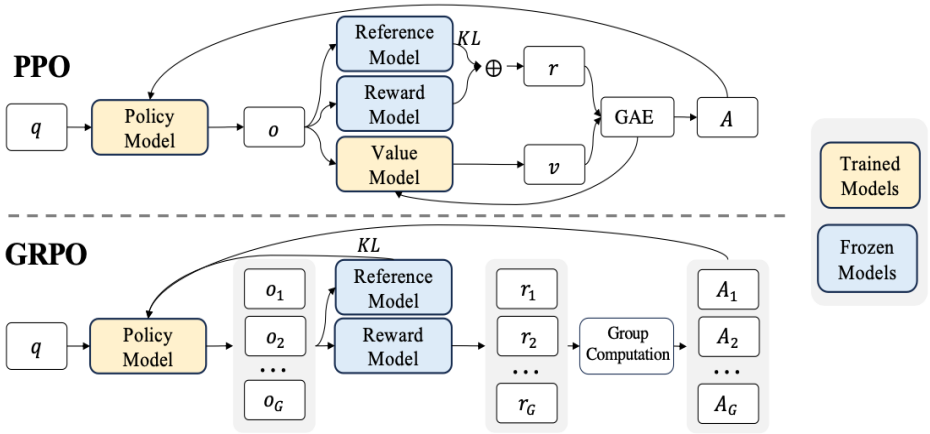
Distributing parallel training computations across heterogeneous GPUs is a natural option to democratize LLM training. In the current exciting era of generative AI, chip vendors typically release new generations of AI chips every 24 months. For instance, Nvidia introduced the Turing architecture in 2018 (Nvidia, 2018), Ampere in 2020 (Nvidia, 2020), Hopper in 2022 (Nvidia, 2022), and Blackwell is scheduled for Q4, 2024 (Nvidia, 2024). On the other hand, one particular version of an AI chip often remains in use by cloud service platforms, technology companies, or research institutions for a much longer period. For example, K80 GPUs with Tesla architecture, released in 2006 (Nvidia, 2006), are still available on AWS as p2 instances (Amazon, 2024). This observation highlights the important opportunity to explore effective ways to maximize efficiency of such widely available yet heterogeneous hardware to facilitate more cost-effective and accessible LLM training.

On the other hand, deploying large-scale training computation for LLM over a set of heterogeneous GPUs with different technique specs would be a challenging task regarding training system design and implementation. To effectively distribute the training computation over thousands of GPUs, the state-of-the-art training systems, like Megatron (Narayanan et al., 2021b) and DeepSpeed (Rajbhandari et al., 2020) usually supports: (i) tensor model parallelism (Narayanan et al., 2021b; Nagrecha, 2021); (ii) pipeline parallelism (Huang et al., 2019; Narayanan et al., 2019; Yang et al., 2021; Narayanan et al., 2021a); and (iii)

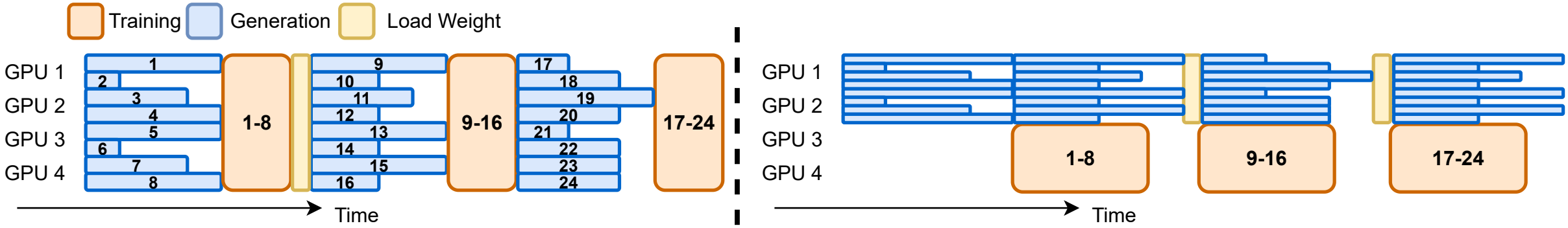
[*]Equal contribution [1]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China [2]School of Computer Science, Peking University, Beijing, China [3]School of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Binhang Yuan <biyuan@ust.hk>.

[MLSys 2026]

RL Post-Training for LLM



- **Rollout generation**: The policy LLM produces responses from a batch of prompts using generative inference.
- **Reward computation**: Using prompts and generated responses, the critic computes their values, the reference policy computes their reference log probabilities, and the reward model computes their rewards, potentially with functions executed in a sandbox.
- **Training**: The policy LLM is updated via gradient-based optimization, using the batch of data produced by previous stages and the reward function.

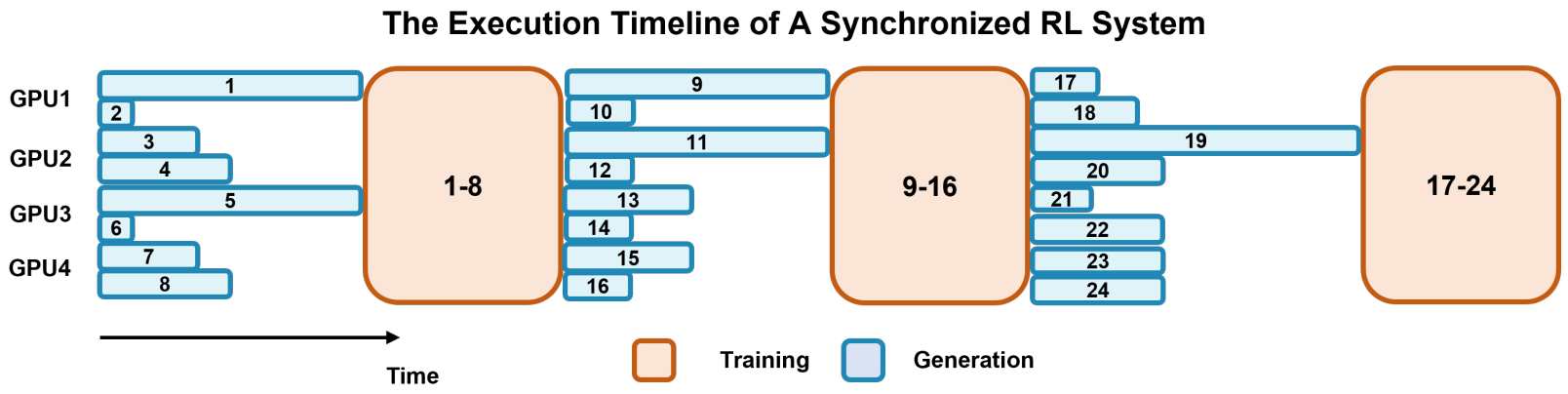


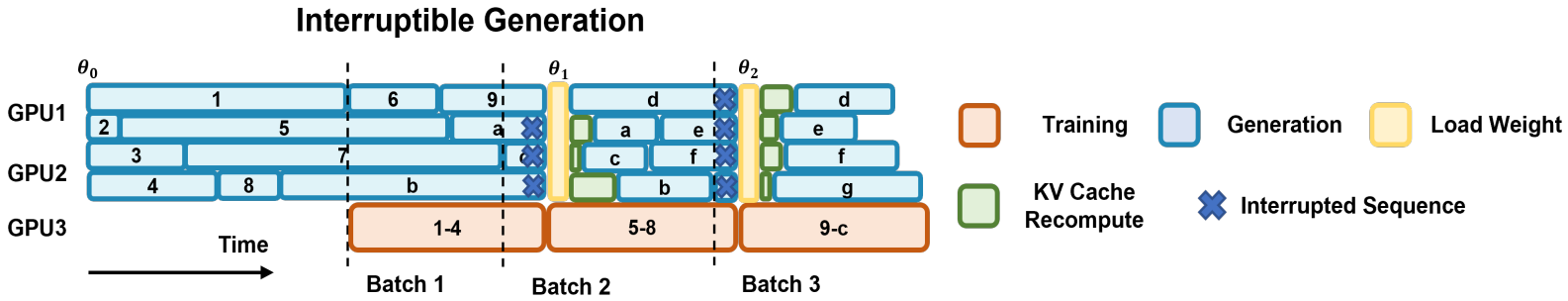
AReaL 1.0

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Asynchronous RL Is Efficient for LLM Post-Training





AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning

Wei Fu[12], Jiaxuan Gao[1], Xujie Shen[2], Chen Zhu[2], Zhiyu Mei[12], Chuyi He[2], Shusheng Xu[12], Guo Wei[2], Jun Mei[2], Jiashu Wang[3], Tongkai Yang[2], Binhang Yuan[3], Yi Wu[1]

[1] IIIS, Tsinghua University, [2] Ant Research, [3] HKUST

fuwth17@gmail.com, jxwuyi@gmail.com

Abstract

Reinforcement learning (RL) has become a trending paradigm for training large language models (LLMs), particularly for reasoning tasks. Effective RL for LLMs requires massive parallelization and poses an urgent need for efficient training systems. Most existing large-scale RL systems for LLMs are synchronous by alternating generation and training in a batch setting, where the rollouts in each training batch are generated by the same (or latest) model. This stabilizes RL training but suffers from severe system-level inefficiency. Generation must wait until the longest output in the batch is completed before model update, resulting in GPU underutilization. We present AReaL, a *fully asynchronous* RL system that completely decouples generation from training. Rollout workers in AReaL continuously generate new outputs without waiting, while training workers update the model whenever a batch of data is collected. AReaL also incorporates a collection of system-level optimizations, leading to substantially higher GPU utilization. To stabilize RL training, AReaL balances the workload of rollout and training workers to control data staleness, and adopts a staleness-enhanced PPO variant to better handle outdated training samples. Extensive experiments on math and code reasoning benchmarks show that AReaL achieves **up to 2.77× training speedup** compared to synchronous systems with the same number of GPUs and matched or even improved final performance. The code of AReaL is available at https://github.com/inclusionAI/AReaL/.

1 Introduction

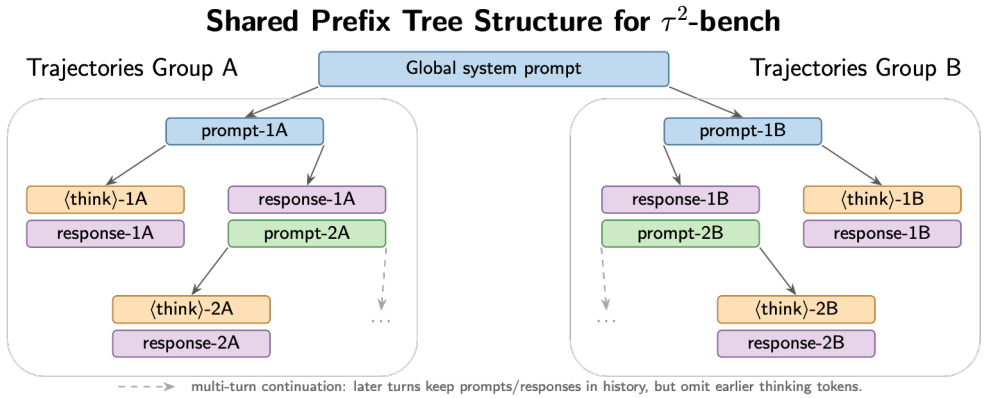
Reinforcement learning (RL) has emerged as a new scaling paradigm for enhancing the capabilities of large language models (LLMs) by enabling thinking abilities [52]. Given a prompt, RL allows an LLM to generate thinking tokens before outputting a final answer, enabling *test-time scaling* [29, 47]. These thinking LLMs are named Large Reasoning Models (LRMs) and have been shown to have particularly strong capabilities on challenging reasoning problems, such as math [9, 5, 20], coding [3, 14, 15], logic puzzles [22, 34], and agentic tasks [23, 57].

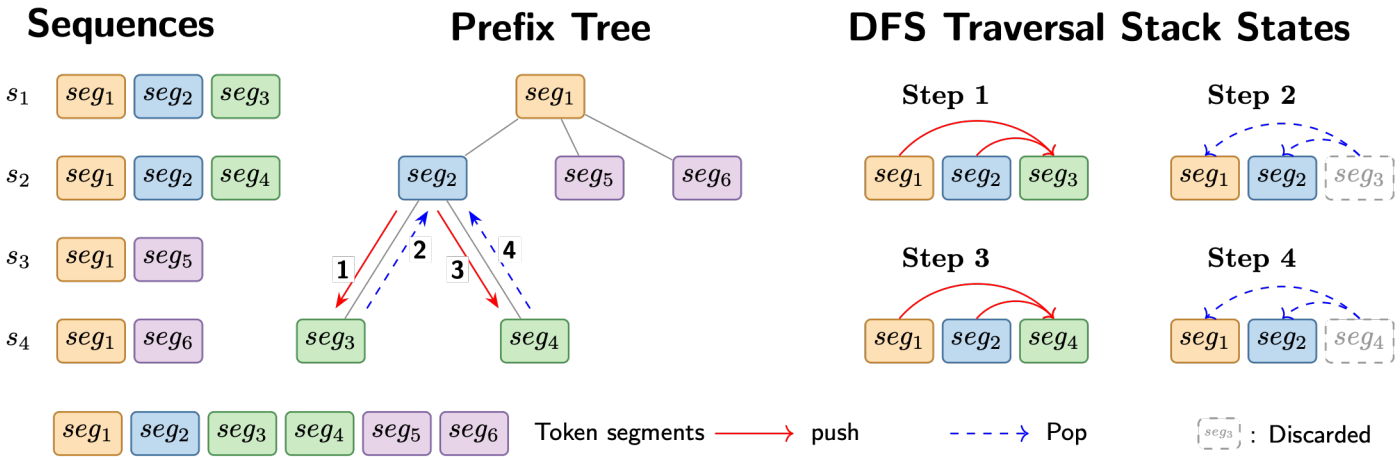
Effective RL training often requires massive parallelization to derive a large batch of rollouts for sufficient exploration, which is the key to obtaining optimal model performance. For example, popular RL algorithms, such as PPO [42] and GRPO [43], often require an effective training batch of thousands of outputs [60, 61, 53]. Moreover, an LRM can generate tens of thousands of thinking tokens for each input prompt [6], further posing an urgent need for an efficient training system to run RL training on a large scale.

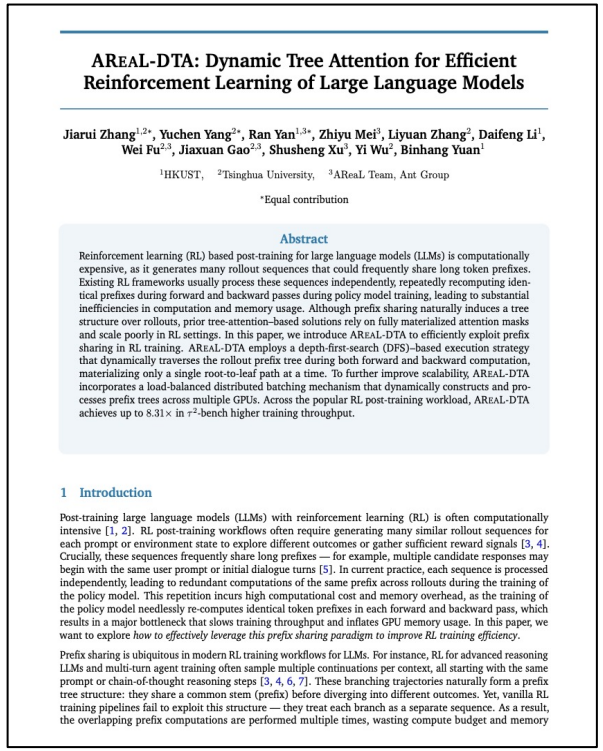
[NeurIPS 2025]

System Research on AReaL: AReaL-DTA

- Reuses shared rollout prefixes to avoid redundant computation.
- Traverses prefix trees dynamically with a DFS-based attention strategy.
- Balances prefix-tree workloads across multiple GPUs.







AReaL-DTA: Dynamic Tree Attention for Efficient Reinforcement Learning of Large Language Models

Jiarui Zhang[1,2*], Yuchen Yang[2*], Ran Yan[1,3*], Zhiyu Mei[3], Liyuan Zhang[2], Daifeng Li[1], Wei Fu[2,3], Jiaxuan Gao[2,3], Shusheng Xu[3], Yi Wu[2], Binhang Yuan[1]

[1]HKUST, [2]Tsinghua University, [3]AReaL Team, Ant Group

*Equal contribution

Abstract

Reinforcement learning (RL) based post-training for large language models (LLMs) is computationally expensive, as it generates many rollout sequences that could frequently share long token prefixes. Existing RL frameworks usually process these sequences independently, repeatedly recomputing identical prefixes during forward and backward passes during policy model training, leading to substantial inefficiencies in computation and memory usage. Although prefix sharing naturally induces a tree structure over rollouts, prior tree-attention–based solutions rely on fully materialized attention masks and scale poorly in RL settings. In this paper, we introduce AReaL-DTA to efficiently exploit prefix sharing in RL training. AReaL-DTA employs a depth-first-search (DFS)–based execution strategy that dynamically traverses the rollout prefix tree during both forward and backward computation, materializing only a single root-to-leaf path at a time. To further improve scalability, AReaL-DTA incorporates a load-balanced distributed batching mechanism that dynamically constructs and processes prefix trees across multiple GPUs. Across the popular RL post-training workload, AReaL-DTA achieves up to 8.31× in τ^2-bench higher training throughput.

1 Introduction

Post-training large language models (LLMs) with reinforcement learning (RL) is often computationally intensive [1, 2]. RL post-training workflows often require generating many similar rollout sequences for each prompt or environment state to explore different outcomes or gather sufficient reward signals [3, 4]. Crucially, these sequences frequently share long prefixes — for example, multiple candidate responses may begin with the same user prompt or initial dialogue turns [5]. In current practice, each sequence is processed independently, leading to redundant computations of the same prefix across rollouts during the training of the policy model. This repetition incurs high computational cost and memory overhead, as the training of the policy model needlessly re-computes identical token prefixes in each forward and backward pass, which results in a major bottleneck that slows training throughput and inflates GPU memory usage. In this paper, we want to explore *how to effectively leverage this prefix sharing paradigm to improve RL training efficiency*.

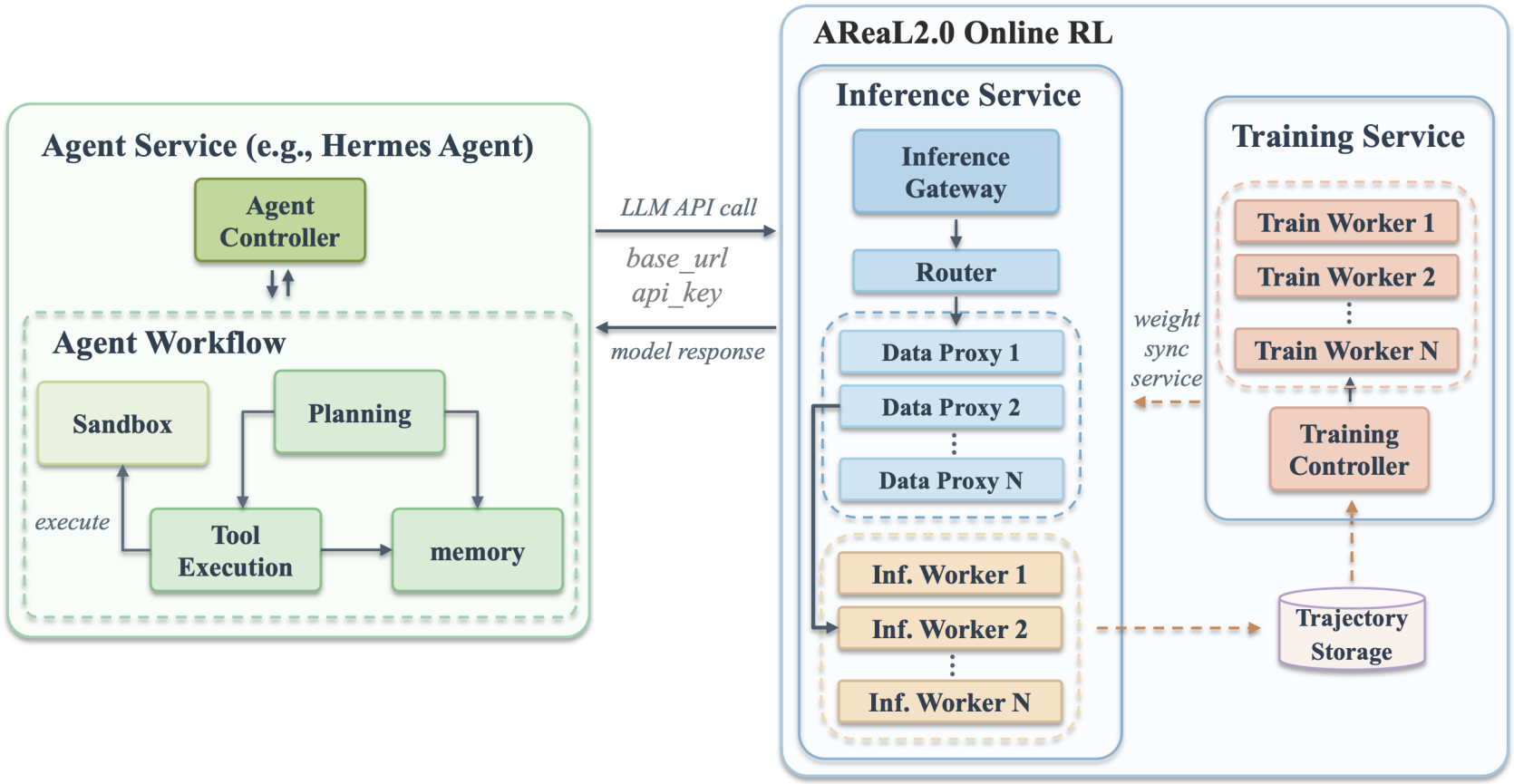
Prefix sharing is ubiquitous in modern RL training workflows for LLMs. For instance, RL for advanced reasoning LLMs and multi-turn agent training often sample multiple continuations per context, all starting with the same prompt or chain-of-thought reasoning steps [3, 4, 6, 7]. These branching trajectories naturally form a prefix tree structure: they share a common stem (prefix) before diverging into different outcomes. Yet, vanilla RL training pipelines fail to exploit this structure — they treat each branch as a separate sequence. As a result, the overlapping prefix computations are performed multiple times, wasting compute budget and memory

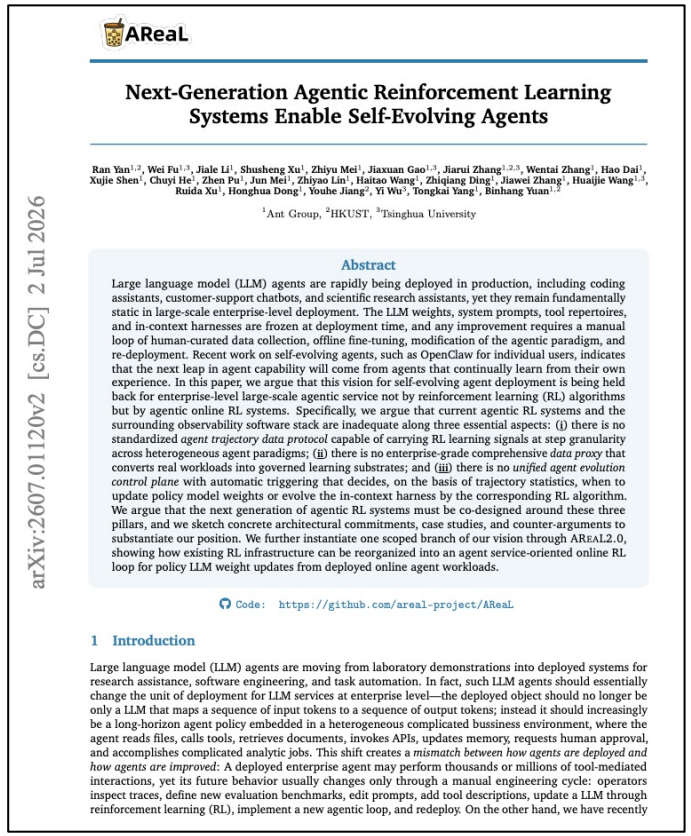
[ICML 2026]

AReaL 2.0: Agentic Online RL Infra

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB





AReaL

Next-Generation Agentic Reinforcement Learning Systems Enable Self-Evolving Agents

Ran Yan[1,2], Wei Fu[1,3], Jiale Li[1], Shusheng Xu[1], Zhiyu Mei[1], Jiaxuan Gao[1,3], Jiarui Zhang[1,2,3], Wentai Zhang[1], Hao Dai[1], Xujie Shen[1], Chuyi He[1], Zhen Pu[1], Jun Mei[1], Zhiyao Lin[1], Haitao Wang[1], Zhiqiang Ding[1], Jiawei Zhang[1], Huaijie Wang[1,3], Ruida Xu[1], Honghua Dong[1], Youhe Jiang[2], Yi Wu[3], Tongkai Yang[1], Binhang Yuan[1,2]

[1]Ant Group, [2]HKUST, [3]Tsinghua University

Abstract

Large language model (LLM) agents are rapidly being deployed in production, including coding assistants, customer-support chatbots, and scientific research assistants, yet they remain fundamentally static in large-scale enterprise-level deployment. The LLM weights, system prompts, tool repertoires, and in-context harnesses are frozen at deployment time, and any improvement requires a manual loop of human-curated data collection, offline fine-tuning, modification of the agentic paradigm, and re-deployment. Recent work on self-evolving agents, such as OpenClaw for individual users, indicates that the next leap in agent capability will come from agents that continually learn from their own experience. In this paper, we argue that this vision for self-evolving agent deployment is being held back for enterprise-level large-scale agentic service not by reinforcement learning (RL) algorithms but by agentic online RL systems. Specifically, we argue that current agentic RL systems and the surrounding observability software stack are inadequate along three essential aspects: (i) there is no standardized *agent trajectory data protocol* capable of carrying RL learning signals at step granularity across heterogeneous agent paradigms; (ii) there is no enterprise-grade comprehensive *data proxy* that converts real workloads into governed learning substrates; and (iii) there is no *unified agent evolution control plane* with automatic triggering that decides, on the basis of trajectory statistics, when to update policy model weights or evolve the in-context harness by the corresponding RL algorithm. We argue that the next generation of agentic RL systems must be co-designed around these three pillars, and we sketch concrete architectural commitments, case studies, and counter-arguments to substantiate our position. We further instantiate one scoped branch of our vision through AReaL2.0, showing how existing RL infrastructure can be reorganized into an agent service-oriented online RL loop for policy LLM weight updates from deployed online agent workloads.

Code: https://github.com/areal-project/AReaL

1 Introduction

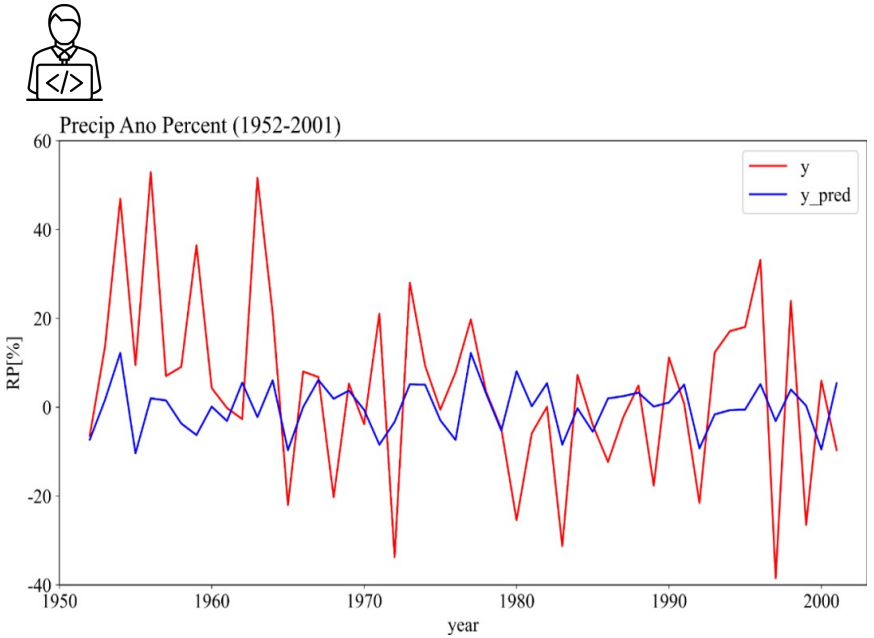
Large language model (LLM) agents are moving from laboratory demonstrations into deployed systems for research assistance, software engineering, and task automation. In fact, such LLM agents should essentially change the unit of deployment for LLM services at enterprise level—the deployed object should no longer be only a LLM that maps a sequence of input tokens to a sequence of output tokens; instead it should increasingly be a long-horizon agent policy embedded in a heterogeneous complicated bussiness environment, where the agent reads files, calls tools, retrieves documents, invokes APIs, updates memory, requests human approval, and accomplishes complicated analytic jobs. This shift creates a *mismatch between how agents are deployed and how agents are improved*: A deployed enterprise agent may perform thousands or millions of tool-mediated interactions, yet its future behavior usually changes only through a manual engineering cycle: operators inspect traces, define new evaluation benchmarks, edit prompts, add tool descriptions, update a LLM through reinforcement learning (RL), implement a new agentic loop, and redeploy. On the other hand, we have recently

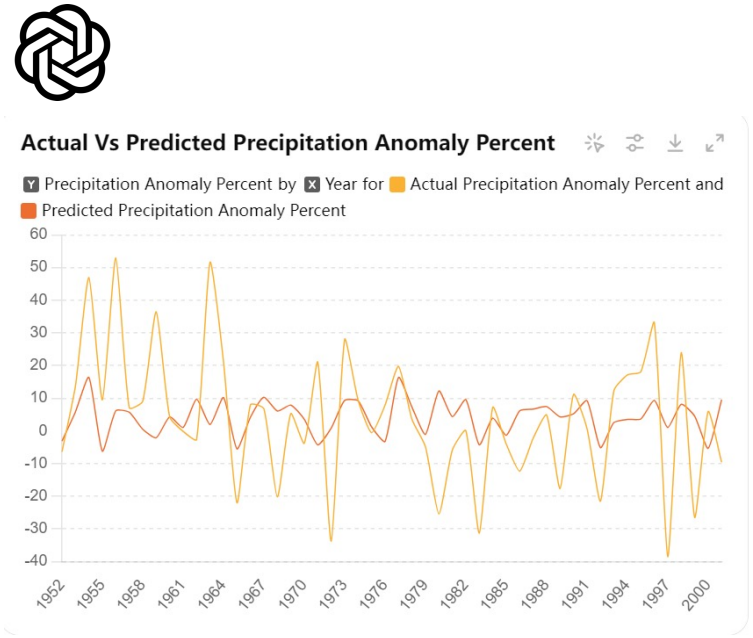
[Arxiv 2607.01120]

LLM Evaluation

How Powerful is LLM for Atmospheric Science? - A Case Study

The attached file is several meteorological variables in different years. The y is Precipitation Anomaly percent in the second column, the other five columns are five x meteorological variables to predict y. Develop a statistical model using Multiple Linear Regression between y and x1 x2 x3 x4 x5, display the figure showing y and y_predict legends calculated by model.





AGU ADVANCING EARTH AND SPACE SCIENCES

Water Resources Research

COMMENTARY
10.1029/2024WR039553

Foundation Models as Assistive Tools in Hydrometeorology: Opportunities, Challenges, and Perspectives

Key Points:
- Foundation models are valuable tools for assisting humans in many tasks, including data processing, weather diagnosis, and decision-making
- Concerns such as hallucinations and responsibility in the use of foundation models emphasize the need for careful implementation
- Enhanced human-AI collaboration and the development of domain-specific foundation models are key in advancing the future of hydrometeorology

Supporting Information:
Supporting Information may be found in the online version of this article.

Correspondence to:
M. Lu,
mengqian.lu@ust.hk

Citation:
Zhang, L., Song, Y., Cui, H., Lu, M., Li, C., Yuan, B., et al. (2025). Foundation models as assistive tools in hydrometeorology: Opportunities, challenges, and perspectives. *Water Resources Research*, *61*, e2024WR039553. https://doi.org/10.1029/2024WR039553

Received 26 NOV 2024
Accepted 7 APR 2025

Author Contributions:
Conceptualization: Mengqian Lu, Binhang Yuan
Formal analysis: Lujia Zhang, Yurong Song, Hanzhe Cui
Funding acquisition: Mengqian Lu
Investigation: Lujia Zhang, Yurong Song, Hanzhe Cui, Mengqian Lu, Bin Wang, Upmanu Lall, Jing Yang
Methodology: Lujia Zhang, Binhang Yuan
Project administration: Mengqian Lu
Software: Lujia Zhang, Yurong Song, Hanzhe Cui
Supervision: Mengqian Lu

Lujia Zhang[1,2], Yurong Song[1], Hanzhe Cui[1], Mengqian Lu[1], Chenyue Li[3], Binhang Yuan[3], Bin Wang[4], Upmanu Lall[5], and Jing Yang[6]

[1]Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong, China, [2]Individual Interdisciplinary Program (Earth, Ocean and Atmospheric Science), Interdisciplinary Program Office, The Hong Kong University of Science and Technology, Hong Kong, China, [3]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China, [4]Department of Atmospheric Sciences and International Pacific Research Center, University of Hawai'i at Mānoa, Honolulu, HI, USA, [5]School of Complex Adaptive Systems, & Water Institute, Arizona State University, Tempe, AZ, USA, [6]State Key Laboratory of Earth Surface Processes and Hazards Risk Governance, Faculty of Geographical Science, Beijing Normal University, Beijing, China

Abstract Most state-of-the-art AI applications in hydrometeorology are based on classic deep learning approaches. However, such approaches cannot automatically integrate multiple functions to construct a single intelligent agent, as each function is enabled by a separate model trained on independent data sets. Foundation models (FMs), which can process diverse inputs and perform different tasks, present a substantial opportunity to overcome this challenge. In this commentary, we evaluate how three state-of-the-art FMs, specifically GPT-4o, Claude 3.5 Sonnet, and Gemini 1.5 Pro, perform across four key task types in hydrometeorology: data processing, event diagnosis, forecast and prediction, and decision-making. The models perform well in the first two task types and offer valuable information for decision-makers but still face challenges in generating reliable forecasts. Moreover, this commentary highlights the concerns regarding the use of FMs: hallucination, responsibility, over-reliance, and openness. Finally, we propose that enhancing human-AI collaboration and developing domain-specific FMs could drive the future of FM applications in hydrometeorology. We also provide specific recommendations to achieve the perspectives.

1. Introduction

The application of artificial intelligence (AI) techniques is driving exciting and promising advancements in uncovering the complexities of our Earth (Sun et al., 2022). Classical machine learning methods, including artificial neural networks, support vector machines, and random forests, have demonstrated remarkable success in various classification and regression tasks in geoscience (Toms et al., 2020; White et al., 2023; Xie et al., 2024). More recently, deep learning methods have achieved substantial success in forecasting across various timescales, including precipitation nowcasting (Y. Zhang, Long, et al., 2023), short- to medium-range weather prediction (Bi et al., 2023; Lam et al., 2023), long-term climate signal forecasting (Ham et al., 2019; Zhou & Zhang, 2023). Beyond purely statistical approaches, AI techniques can be integrated with established domain-specific methods such as numerical simulations (Kochkov et al., 2024), or hydrological models (Xie et al., 2024) to develop physics-informed models. These advances illustrate the transformative role of AI in hydrometeorology, demonstrating significant contributions to data management, modeling, prediction, and process understanding while promising even greater advances in the future (M. Chen et al., 2024).

Despite their successes, these models are typically designed for specific tasks with customized inputs and outputs, raising concerns about the generalizability of AI methods to diverse real-world applications, particularly under constraints of limited data and computational resources (Nguyen et al., 2023). Moreover, training on a single data set or limited data types restricts models to learning only a narrow set of explicit or implicit properties due to potential data homogeneity (Sun et al., 2022), whereas an ideal AI system should be able to process heterogeneous inputs and integrate sophisticated reasoning procedures. Such requirements go beyond the scope of classic supervised or semi-supervised machine learning techniques—addressing these challenges calls for the application of more advanced AI techniques that can dynamically adapt to diverse data types and execute complex, multi-step analytical tasks, thereby enhancing the robustness and applicability of AI in real-world hydrometeorological scenarios (Aghajanyan et al., 2022; Lu et al., 2022).

[WRR 2025]

LLM Evaluation

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

How Powerful is LLM for Atmospheric Science? - A Comprehensive Benchmark

Category	Model	Hydro	AtmDyn	AtmosPhy	GeoPhy	PhyOcean	Overall Acc	SymStd.
Instruction Models	Gemma-2-9B-it	24.0	13.78	15.71	11.43	20.0	15.08	4.07
	Gemma-2-27B-it	56.0	29.73	45.71	41.43	37.5	36.72	5.94
	Qwen2.5-3B-Instruct	46.0	29.19	34.28	30.0	37.5	32.09	7.71
	Qwen2.5-7B-Instruct	62.0	38.92	51.43	51.43	42.5	44.78	5.12
	Qwen2.5-32B-Instruct	58.0	47.3	64.28	62.86	55.0	53.73	6.05
	Qwen2.5-72B-Instruct-Turbo	72.0	50.0	77.86	44.29	62.5	57.61	4.73
	Llama-3.3-70B-Instruct	78.0	42.94	67.14	51.91	52.5	52.11	3.73
	Llama-3.1-405B-Instruct-Turbo	72.0	48.92	64.29	57.14	62.5	55.52	6.17
	GPT-4o-mini	48.0	42.16	58.57	40.0	40.0	45.67	4.78
	GPT-4o	68.0	51.08	74.28	60.0	55.0	58.36	5.19
	Gemini-2.0-Flash-Exp	90.0	58.11	68.57	77.14	62.5	64.93	4.29
	Deepseek-V3	94.0	57.3	73.57	64.28	62.5	64.48	6.28
Reasoning Models	QwQ-32B-Preview	88.0	60.27	87.86	74.28	60.0	69.55	4.57
	Gemini-2.0-Flash-Thinking-Exp (01-21)	100.0	78.11	85.0	91.43	80.0	82.69	3.87
	GPT-o1	100.0	82.7	92.14	92.86	87.5	87.31	3.32
	Deepseek-R1	98.0	85.68	95.0	95.71	82.5	89.4	3.48
Math Models	Deepseek-Math-7B-RL	22.0	20.54	27.86	24.29	37.5	23.58	4.44
	Deepseek-Math-7B-Instruct	36.0	28.38	33.57	30.0	40.0	30.9	4.04
	Qwen2.5-Math-1.5B-Instruct	50.0	30.0	22.86	34.29	30.0	30.45	3.24
	Qwen2.5-Math-7B-Instruct	54.0	31.35	39.28	35.71	32.5	35.22	6.14
	Qwen2.5-Math-72B-Instruct	70.0	54.87	73.57	62.86	40.0	59.85	6.27
Domain-Specific Models	ClimateGPT-7B	26.0	18.65	21.43	11.43	30.0	19.7	5.25
	ClimateGPT-70B	24.0	25.4	30.0	40.0	27.5	27.91	4.45

ATMOSSCI-BENCH: Evaluating the Recent Advances of Large Language Models for Atmospheric Science

Chenyue Li
CSE, HKUST
Clear Water Bay, Hong Kong SAR
clieh@connect.ust.hk

Wen Deng
CE, HKUST
Clear Water Bay, Hong Kong SAR
wdengan@connect.ust.hk

Mengqian Lu
CE, HKUST
Clear Water Bay, Hong Kong SAR
cemlu@ust.hk

Binhang Yuan*
CSE, HKUST
Clear Water Bay, Hong Kong SAR
biyuan@ust.hk

Abstract

The rapid advancements in large language models (LLMs), particularly in their reasoning capabilities, hold transformative potential for addressing complex challenges and boosting scientific discovery in atmospheric science. However, leveraging LLMs effectively in this domain requires a robust and comprehensive evaluation benchmark. Toward this end, we present ATMOSSCI-BENCH, a novel benchmark designed to systematically assess LLM performance across five core categories of atmospheric science problems: hydrology, atmospheric dynamics, atmospheric physics, geophysics, and physical oceanography. ATMOSSCI-BENCH features a dual-format design comprising both multiple-choice questions (MCQs) and open-ended questions (OEQs), enabling scalable automated evaluation alongside deeper analysis of conceptual understanding. We employ a template-based MCQ generation framework to create diverse, graduate-level problems with symbolic perturbation, while OEQs are used to probe open-ended reasoning. We conduct a comprehensive evaluation of representative LLMs, categorized into four groups: instruction-tuned models, advanced reasoning models, math-augmented models, and domain-specific climate models. Our analysis provides some interesting insights into the reasoning and problem-solving capabilities of LLMs in atmospheric science. We believe ATMOSSCI-BENCH can serve as a critical step toward advancing LLM applications in climate services by offering a standard and rigorous evaluation framework. The source code of ATMOSSCI-BENCH is available at [https://github.com/Relaxed-System-Lab/AtmosSci-Bench].

1 Introduction

Large language models (LLMs) [1], especially in their reasoning capabilities, have recently achieved remarkable progress, offering transformative potential for addressing complex challenges in atmospheric science [2, 3, 4, 5]. More recently, increasingly powerful LLMs have accelerated progress in AI4S (AI for Science), enabling a paradigm shift in scientific discovery. With their growing capabilities, LLMs show the potential to act as "AI Scientists," partially assisting—or even autonomously conducting—hypothesis generation, experimental design, execution, analysis, and refinement [6, 7, 8, 9, 10, 11]. To advance AI for Atmospheric Science and enable the development of reliable and effective LLM-based applications for climate-related tasks, it is crucial to recognize

*Corresponding author.

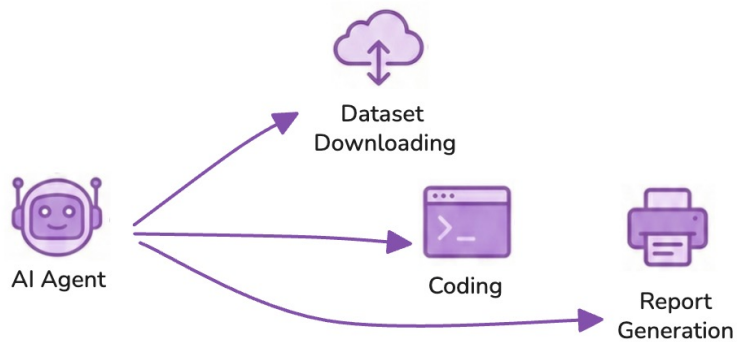
39th Conference on Neural Information Processing Systems (NeurIPS 2025).

[NeurIPS 2025]

LLM Evaluation

How Powerful is LLM for Atmospheric Science?
- LLM Climate Agent Orchestration

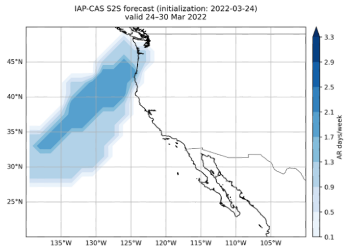
Produce the AR (atmospheric river) frequency forecast map for the week of March 24–30, 2022, using the IAP-CAS (origin) S2S database for the West America region (covering approximately 20°N–50°N latitude and 100°W–140°W longitude)



End User

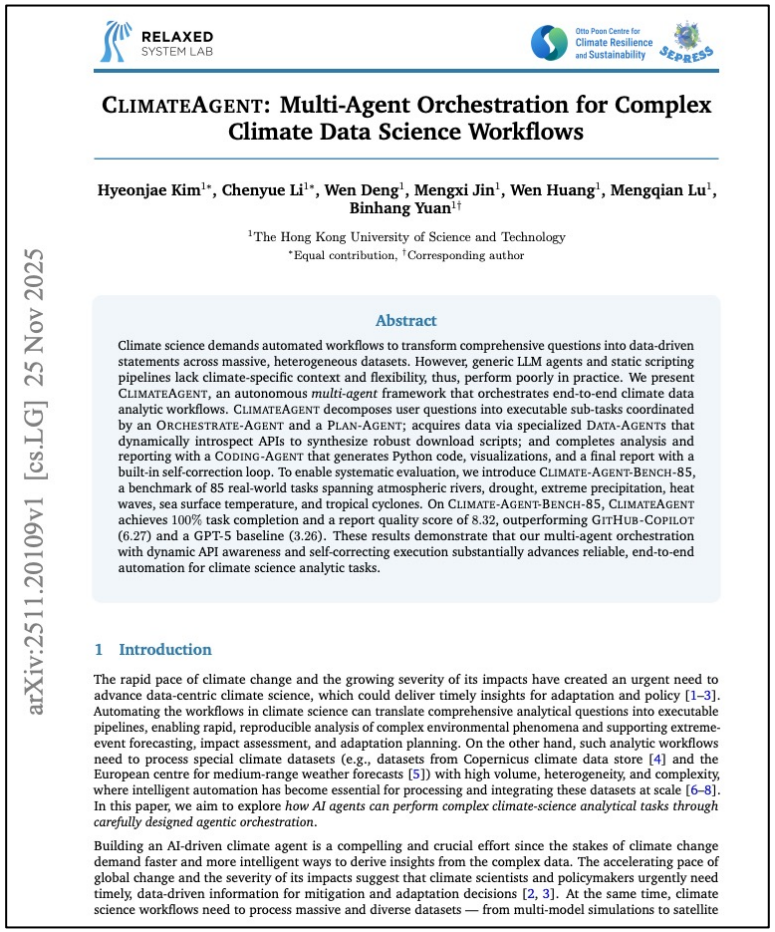
- Autonomously download specific dataset.
- Autonomously write script for visualization.
- Autonomously generate report.

Atmospheric River (AR) Frequency Forecast



Data source: IAP-CAS Subseasonal-to-Seasonal (S2S) forecast database
Initialization date:** 24 Mar 2022
Forecast validity period: 24–30 Mar 2022
Spatial domain: West America region (20°N–50°N, 100°W–140°W)

Main Spatial Patterns (AR Frequency):
- **Northern California (38.5–42°N):** 0.04 (mean), peaking locally up to 1.00 AR days/week.
- **Central California (35–38.5°N):** 0.00 (mean), with maxima of 0.00 AR days/week.
- **Southern California (32.5–35°N):** 0.00 (mean), reaching up to 0.00 AR days/week in select areas.
- **Offshore (Pacific, 32–45°N, 140–126°W):** 0.92 (mean), with enhanced AR frequency patches up to 2.00 AR days/week.



RELAXED SYSTEM LAB

Otto Poon Centre for Climate Resilience and Sustainability

CLIMATEAGENT: Multi-Agent Orchestration for Complex Climate Data Science Workflows

Hyeonjae Kim[1*], Chenyue Li[1*], Wen Deng[1], Mengxi Jin[1], Wen Huang[1], Mengqian Lu[1], Binhang Yuan[1†]

[1]The Hong Kong University of Science and Technology
*Equal contribution, †Corresponding author

Abstract

Climate science demands automated workflows to transform comprehensive questions into data-driven statements across massive, heterogeneous datasets. However, generic LLM agents and static scripting pipelines lack climate-specific context and flexibility, thus, perform poorly in practice. We present CLIMATEAGENT, an autonomous *multi-agent* framework that orchestrates end-to-end climate data analytic workflows. CLIMATEAGENT decomposes user questions into executable sub-tasks coordinated by an ORCHESTRATE-AGENT and a PLAN-AGENT; acquires data via specialized DATA-AGENTS that dynamically introspect APIs to synthesize robust download scripts; and completes analysis and reporting with a CODING-AGENT that generates Python code, visualizations, and a final report with a built-in self-correction loop. To enable systematic evaluation, we introduce CLIMATE-AGENT-BENCH-85, a benchmark of 85 real-world tasks spanning atmospheric rivers, drought, extreme precipitation, heat waves, sea surface temperature, and tropical cyclones. On CLIMATE-AGENT-BENCH-85, CLIMATEAGENT achieves 100% task completion and a report quality score of 8.32, outperforming GITHUB-COPILOT (6.27) and a GPT-5 baseline (3.26). These results demonstrate that our multi-agent orchestration with dynamic API awareness and self-correcting execution substantially advances reliable, end-to-end automation for climate science analytic tasks.

1 Introduction

The rapid pace of climate change and the growing severity of its impacts have created an urgent need to advance data-centric climate science, which could deliver timely insights for adaptation and policy [1–3]. Automating the workflows in climate science can translate comprehensive analytical questions into executable pipelines, enabling rapid, reproducible analysis of complex environmental phenomena and supporting extreme-event forecasting, impact assessment, and adaptation planning. On the other hand, such analytic workflows need to process special climate datasets (e.g., datasets from Copernicus climate data store [4] and the European centre for medium-range weather forecasts [5]) with high volume, heterogeneity, and complexity, where intelligent automation has become essential for processing and integrating these datasets at scale [6–8]. In this paper, we aim to explore *how AI agents can perform complex climate-science analytical tasks through carefully designed agentic orchestration*.

Building an AI-driven climate agent is a compelling and crucial effort since the stakes of climate change demand faster and more intelligent ways to derive insights from the complex data. The accelerating pace of global change and the severity of its impacts suggest that climate scientists and policymakers urgently need timely, data-driven information for mitigation and adaptation decisions [2, 3]. At the same time, climate science workflows need to process massive and diverse datasets — from multi-model simulations to satellite

[TMLR 2026]

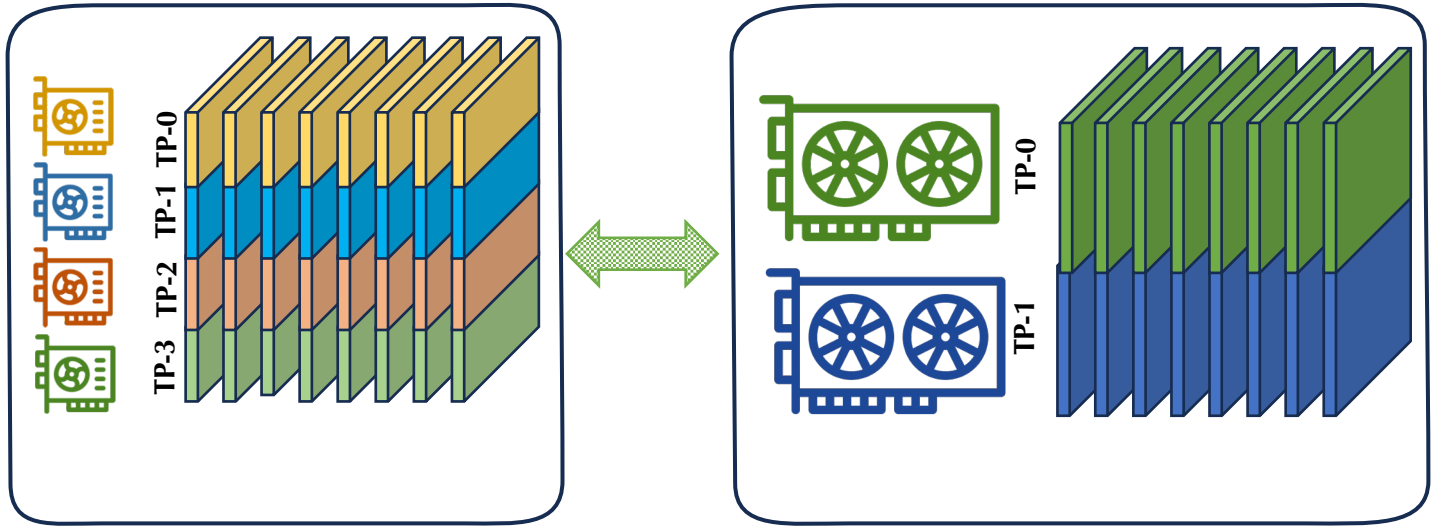
Heterogeneous Inference Serving: HexGen

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Co-locating Prefill and Decoding:

- An implementation that accommodates tensor model parallelism and pipeline parallelism.
- A scheduling algorithm that optimizes pipeline partitions and parallel strategies over heterogeneous GPUs:
  - optimizing the layout of a pipeline through dynamic programming;
  - solving the global scheduling through a genetic algorithm.



HEXGEN: Generative Inference of Large Language Model over Heterogeneous Environment

Youhe Jiang [*1] Ran Yan [*1] Xiaozhe Yao [*2] Yang Zhou [3] Beidi Chen [3] Binhang Yuan [1]

Abstract

Serving generative inference of the large language model is a crucial component of contemporary AI applications. This paper focuses on deploying such services in a heterogeneous and cross-datacenter setting to mitigate the substantial inference costs typically associated with a single centralized datacenter. Towards this end, we propose HEXGEN, a *flexible distributed inference engine* that uniquely supports the asymmetric partition of generative inference computations over both tensor model parallelism and pipeline parallelism and allows for effective deployment across diverse GPUs interconnected by a fully heterogeneous network. We further propose a *sophisticated scheduling algorithm* grounded in constrained optimization that can adaptively assign asymmetric inference computation across the GPUs to fulfill inference requests while maintaining acceptable latency levels. We conduct an extensive evaluation to verify the efficiency of HEXGEN by serving the state-of-the-art LLAMA-2 (70B) model. The results suggest that HEXGEN can choose to achieve up to 2.3× lower latency deadlines or tolerate up to 4× more request rates compared with the homogeneous baseline given the same budget. Our implementation is available at https://github.com/Relaxed-System-Lab/HexGen.

1. Introduction

Large language models are distinguished by the vast scale of parameters being trained over a substantial pre-train corpus. Such extensive training enables them to be remarkably adaptable across a broad spectrum of downstream tasks (Bommasani et al., 2021). In fact, large language models such as GPT-4 (Bubeck et al., 2023), Llama2-70B (Touvron et al., 2023), and Falcon-180B (Institute, 2023) have essentially revolutionized the way AI systems are developed and deployed, which have nourished a large number of advanced applications. In such an ecosystem, serving the generative inference requests for large language models presents a critical challenge — given the unprecedented model scale, unlike classic machine learning models, parallel inference strategies have to be leveraged to accommodate the high computational and memory demands while ensuring low-latency generative inference outcomes.

The state-of-the-art inference service of the large language model is usually hosted in a single centralized data center with homogeneous high-performance GPUs, which can be very expensive in terms of the cloud service fee. The high cost of such deployment potentially limits the democratization of this great technique. Alternatively, the deployment of the large language model inference over a heterogeneous cross-datacenter environment can be a promising direction to reduce the inference cost, which has not been fully explored. The heterogeneous environment for foundation model inference service can encompass a wide range of options, including more affordable cloud services (such as spot instances (Thorpe et al., 2023; Athlur et al., 2022) and serverless computing (Guo et al., 2022)) to even fully decentralized platforms (Yuan et al., 2022; Borzunov et al., 2023) that leverage a diverse set of GPUs contributed by volunteers in an extreme setting.

However, deploying large language model inference across a heterogeneous environment presents some unique challenges. Unlike traditional machine learning models, large language model inference consists of two different phases: a prompt phase that handles a sequence of input tokens at once and a decoding phase where output tokens are generated step-by-step. Additionally, large language models require the adoption of specialized *parallel inference strategies* to effectively distribute the intensive computations across multiple GPUs. The two most commonly employed approaches are *tensor model parallelism* and *pipeline parallelism*. In

[*]Equal contribution [1]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China [2]Department of Computer Science, ETH Zurich, Zürich, Switzerland [3]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania. Correspondence to: Binhang Yuan <biyuan@ust.hk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

[ICML 2024]

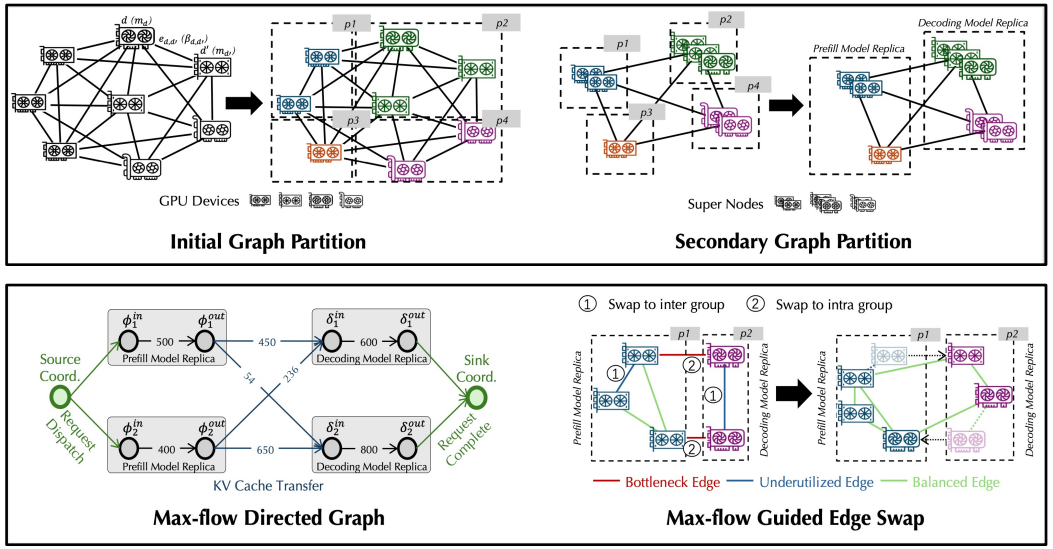
Heterogeneous Inference Serving: HexGen-2

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Prefill–Decoding (PD) Disaggregation:

- Scheduling for the disaggregated framework:
  - **Graph partition**: partition the set of heterogeneous GPUs into multiple model serving groups, where each group could serve a prefill or a decoding phase;
  - **Max-flow**: find the current optimal parallel strategies for prefill and decoding model replicas and generate the optimal KV cache communication strategy among them;
  - **Iterative refinement**: iteratively repeat the two-phase algorithm to find the optimal model placement strategy.



HEXGEN-2: DISAGGREGATED GENERATIVE INFERENCE OF LLMS IN HETEROGENEOUS ENVIRONMENT

Youhe Jiang*, Ran Yan*, Binhang Yuan
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
youhejiang@gmail.com, ryanaf@connect.ust.hk, biyuan@ust.hk

ABSTRACT

Disaggregating the prefill and decoding phases represents an effective new paradigm for generative inference of large language models (LLM), which eliminates prefill-decoding interference and optimizes resource allocation. However, it is still an open problem about how to deploy the disaggregated inference paradigm across a group of heterogeneous GPUs, which can be an economical alternative to deployment over homogeneous high-performance GPUs. Towards this end, we introduce HEXGEN-2, a distributed system for efficient and economical LLM serving on heterogeneous GPUs following the disaggregated paradigm. Built on top of HEXGEN, the core component of HEXGEN-2 is a *scheduling algorithm* that formalizes the allocation of disaggregated LLM inference computations and communications over heterogeneous GPUs and network connections as a constraint optimization problem. We leverage the *graph partitioning* and *max-flow* algorithms to co-optimize resource allocation, parallel strategies for distinct inference phases, and the efficiency of inter-phase key-value (KV) cache communications. We conduct extensive experiments to evaluate HEXGEN-2, i.e., on OPT (30B) and LLAMA-2 (70B) models in various real-world settings, the results reveal that HEXGEN-2 delivers up to a 2.0× and on average a 1.3× improvement in serving throughput, reduces the average inference latency by 1.5× compared with state-of-the-art systems given the same price budget, and achieves comparable inference performance with a 30% lower price budget.

1 INTRODUCTION

Large Language Models (LLMs), such as OPT (Zhang et al., 2022), LLAMA (Touvron et al., 2023), GPT (OpenAI, 2024), GEMINI (Reid et al., 2024), CLAUDE (Anthropic, 2024) and MIXTRAL (Jiang et al., 2024a) have shown exceptional performance across various advanced applications. However, deploying the generative inference service for such LLMs can be costly, typically requiring a substantial number of homogeneous, high-performance GPUs to meet the service demands, such as first token latency and generation throughput. In this paper, we explore an alternative solution that *deploys the most advanced disaggregated generative inference paradigm over a set of heterogeneous GPUs to provide an efficient and economical LLM service*.

Disaggregated inference is currently the most *efficient* framework for serving the generative inference requests of LLMs (Zhong et al., 2024; Patel et al., 2024). By splitting the prefill phase (compute-bounded) and decoding phase (HBM IO-bounded) across different GPUs, the disaggregation significantly reduces interference between different requests and enables more flexible parallel configurations for the two phases. When compared with colocating the prefill and decoding computations, the disaggregated approach optimizes resource usage and enhances the scalability and efficiency of the LLM inference service. Recent efforts (Jiang et al., 2024b; Griggs et al., 2024; Zhao et al., 2024; Miao et al., 2024) have shown that serving LLMs with heterogeneous GPUs can be a *economical* alternative to deploying over homogeneous high-performance GPUs. Heterogeneous deployments offer significant opportunities to reduce inference service costs by leveraging the wide availability of diverse GPU types across commercial and private computing platforms. Note that Nvidia typically releases new GPU generations every 24 months, e.g., Turing in 2018, Ampere in

*Equal contribution

[ICLR 2025]

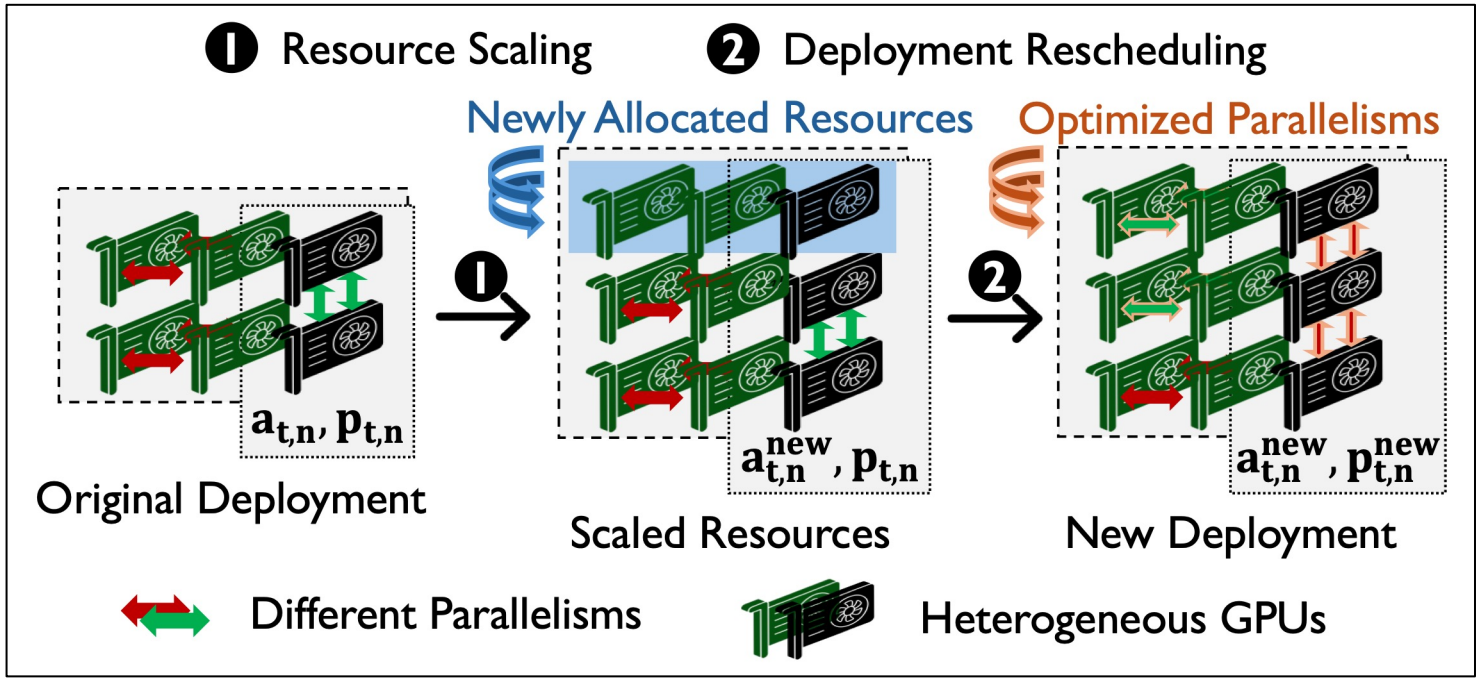
Heterogeneous Inference Serving: HexGen-3

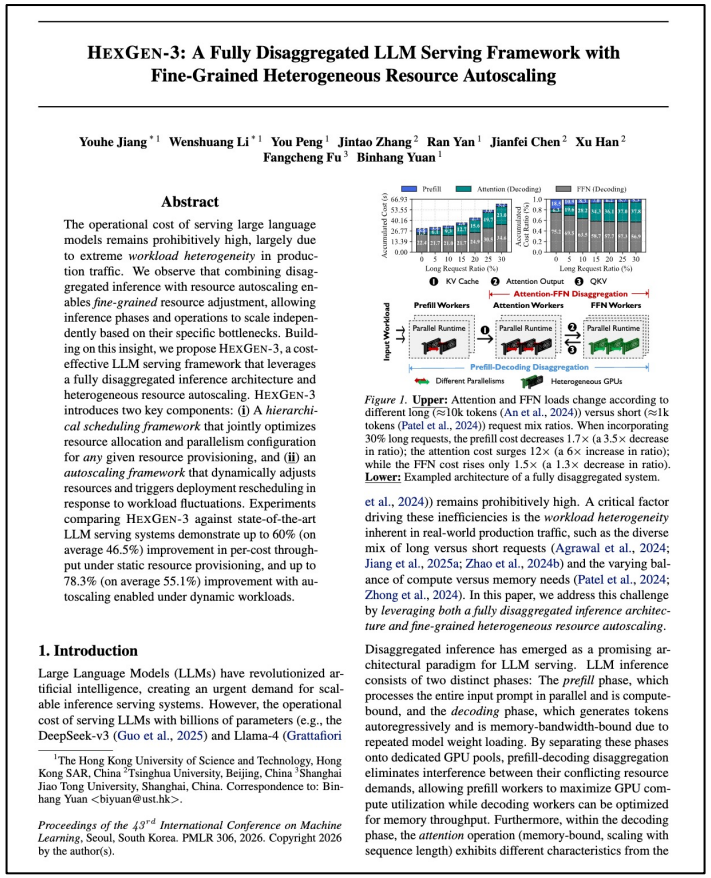
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Attention–FFN (AFD) Disaggregation:

- Fully disaggregate prefill, decode-attention, and decode-FFN into independently scalable workers.
- Jointly optimize heterogeneous GPU allocation and parallelism.
- Dynamically auto-scale resources and reschedules deployments as traffic changes.





HEXGEN-3: A Fully Disaggregated LLM Serving Framework with Fine-Grained Heterogeneous Resource Autoscaling

Youhe Jiang [*1] Wenshuang Li [*1] You Peng [1] Jintao Zhang [2] Ran Yan [1] Jianfei Chen [2] Xu Han [2] Fangcheng Fu [3] Binhang Yuan [1]

Abstract

The operational cost of serving large language models remains prohibitively high, largely due to extreme *workload heterogeneity* in production traffic. We observe that combining disaggregated inference with resource autoscaling enables *fine-grained* resource adjustment, allowing inference phases and operations to scale independently based on their specific bottlenecks. Building on this insight, we propose HEXGEN-3, a cost-effective LLM serving framework that leverages a fully disaggregated inference architecture and heterogeneous resource autoscaling. HEXGEN-3 introduces two key components: (i) A *hierarchical scheduling framework* that jointly optimizes resource allocation and parallelism configuration for *any* given resource provisioning, and (ii) an *autoscaling framework* that dynamically adjusts resources and triggers deployment rescheduling in response to workload fluctuations. Experiments comparing HEXGEN-3 against state-of-the-art LLM serving systems demonstrate up to 60% (on average 46.5%) improvement in per-cost throughput under static resource provisioning, and up to 78.3% (on average 55.1%) improvement with autoscaling enabled under dynamic workloads.

Figure 1. **Upper:** Attention and FFN loads change according to different long (≈10k tokens (An et al., 2024)) versus short (≈1k tokens (Patel et al., 2024)) request mix ratios. When incorporating 30% long requests, the prefill cost decreases 1.7× (a 3.5× decrease in ratio); the attention cost surges 12× (a 6× increase in ratio); while the FFN cost rises only 1.5× (a 1.3× decrease in ratio). **Lower:** Exampled architecture of a fully disaggregated system.

1. Introduction

Large Language Models (LLMs) have revolutionized artificial intelligence, creating an urgent demand for scalable inference serving systems. However, the operational cost of serving LLMs with billions of parameters (e.g., the DeepSeek-v3 (Guo et al., 2025) and Llama-4 (Grattafiori et al., 2024)) remains prohibitively high. A critical factor driving these inefficiencies is the *workload heterogeneity* inherent in real-world production traffic, such as the diverse mix of long versus short requests (Agrawal et al., 2024; Jiang et al., 2025a; Zhao et al., 2024b) and the varying balance of compute versus memory needs (Patel et al., 2024; Zhong et al., 2024). In this paper, we address this challenge by *leveraging both a fully disaggregated inference architecture and fine-grained heterogeneous resource autoscaling.*

Disaggregated inference has emerged as a promising architectural paradigm for LLM serving. LLM inference consists of two distinct phases: The *prefill* phase, which processes the entire input prompt in parallel and is compute-bound, and the *decoding* phase, which generates tokens autoregressively and is memory-bandwidth-bound due to repeated model weight loading. By separating these phases onto dedicated GPU pools, prefill-decoding disaggregation eliminates interference between their conflicting resource demands, allowing prefill workers to maximize GPU compute utilization while decoding workers can be optimized for memory throughput. Furthermore, within the decoding phase, the *attention* operation (memory-bound, scaling with sequence length) exhibits different characteristics from the

[1]The Hong Kong University of Science and Technology, Hong Kong SAR, China [2]Tsinghua University, Beijing, China [3]Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Binhang Yuan <biyuan@ust.hk>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

[ICML 2026]

What is Next?

System for LLM → LLM for System

From Code Assistant to System Engineer

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

> ***Central claim:*** ***LLM agents can act as autonomous system engineers that own design, optimisation, validation, and operation — not merely as code-generation assistants that a human must supervise at every step.***

Conventional: Code assistant

- The human specifies the intent and the design.
- The agent drafts code; the human verifies it.
- Feedback is sparse: the tests pass, or they do not.
- The agent is stateless — nothing accumulates across deployments.
- The human owns the loop, so the loop runs at human speed.

What we want: Autonomous system engineer

- The agent owns design → optimize → validate → operate → adapt.
- The environment supplies dense, structured, verifiable feedback.
- Deterministic machinery — compilers, simulators — guarantees correctness.
- Experience accumulates in a curated, auditable substrate.
- ML system design and implementation could be done by autonomous agents.

Our goal is to: close the engineering loop without a human inside it.

The Infrastructure Bottleneck

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

> ***Observation.*** ***Machine-learning systems are increasingly constrained not by model architecture alone, but by the difficulty of designing, optimising, and continuously adapting the software infrastructure beneath them.***

1 The deployment surface explodes

- Heterogeneous accelerators, shifting workloads, evolving models — each moving on its own schedule.
- The cross-product of hardware × workload × model is not enumerable offline.

2 Hand-engineering cannot keep pace

- Vendor assembly libraries take months of expert effort and trail yearly hardware refreshes.
- Serving policies are tuned per deployment and go stale as conditions drift.

3 Why now

- Frontier coding models make agentic system engineering practical for the first time.
- The missing ingredient is not capability — it is a feedback channel the agent can act on.

The Central Lesson

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

> *Key insight. Autonomous agents become effective system engineers when the environment manufactures dense, structured, verifiable, and timely feedback to guide the optimization.*

ARGUS	**Compile-time data-flow-invariant analysis** Every violation names the offending thread, data element, and program point — a defect report, not a verdict.
Autopoiesis	**Calibrated simulator + per-candidate cost breakdown** Each candidate returns N, Σt_stale, Σt_reconfig, Σt_serve — the agent sees which axis its edit moved.
AReaL-AutoPilot	**Validated deployment plans + runtime telemetry** The plan is checked before launch; execution is monitored and fed back as policy pressure.

A Few Attempts

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

1 ARGUS

Offline kernel synthesis

Agent-generated GPU kernels that reach hand-tuned assembly performance.

Data-flow invariants · compile-time counterexamples · ICRL planner

GEMM · flash attention · MoE, AMD MI300X

2 Autopoiesis

Online inference adaptation

Serving policies that an agent rewrites while the system is live.

Two-plane architecture · LLM-driven program synthesis · hot-swap

Homogeneous · heterogeneous · elastic cloud clusters

3 AReaL-AutoPilot

Full lifecycle for RL tasks (in progress)

An agentic control layer for distributed RL post-training.

Spec → validated plan → launch → monitor → adapt

Built on AReaL; automatic RL deployment.

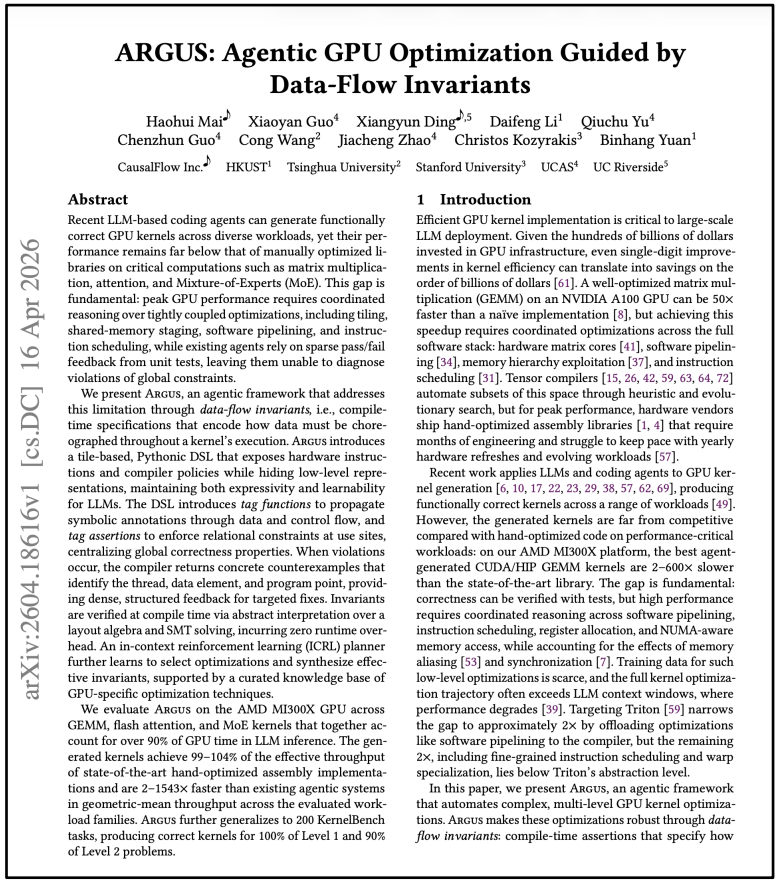
1 ARGUS

Offline kernel synthesis

Agent-generated GPU kernels that reach hand-tuned assembly performance.

Data-flow invariants · compile-time counterexamples · ICRL planner

GEMM · flash attention · MoE, AMD MI300X



ARGUS: Agentic GPU Optimization Guided by Data-Flow Invariants

Haohui Mai[♪] Xiaoyan Guo[4] Xiangyun Ding[♪,5] Daifeng Li[1] Qiuchu Yu[4]
Chenzhun Guo[4] Cong Wang[2] Jiacheng Zhao[4] Christos Kozyrakis[3] Binhang Yuan[1]
CausalFlow Inc.[♪] HKUST[1] Tsinghua University[2] Stanford University[3] UCAS[4] UC Riverside[5]

Abstract

Recent LLM-based coding agents can generate functionally correct GPU kernels across diverse workloads, yet their performance remains far below that of manually optimized libraries on critical computations such as matrix multiplication, attention, and Mixture-of-Experts (MoE). This gap is fundamental: peak GPU performance requires coordinated reasoning over tightly coupled optimizations, including tiling, shared-memory staging, software pipelining, and instruction scheduling, while existing agents rely on sparse pass/fail feedback from unit tests, leaving them unable to diagnose violations of global constraints.

We present Argus, an agentic framework that addresses this limitation through *data-flow invariants*, i.e., compile-time specifications that encode how data must be choreographed throughout a kernel's execution. Argus introduces a tile-based, Pythonic DSL that exposes hardware instructions and compiler policies while hiding low-level representations, maintaining both expressivity and learnability for LLMs. The DSL introduces *tag functions* to propagate symbolic annotations through data and control flow, and *tag assertions* to enforce relational constraints at use sites, centralizing global correctness properties. When violations occur, the compiler returns concrete counterexamples that identify the thread, data element, and program point, providing dense, structured feedback for targeted fixes. Invariants are verified at compile time via abstract interpretation over a layout algebra and SMT solving, incurring zero runtime overhead. An in-context reinforcement learning (ICRL) planner further learns to select optimizations and synthesize effective invariants, supported by a curated knowledge base of GPU-specific optimization techniques.

We evaluate Argus on the AMD MI300X GPU across GEMM, flash attention, and MoE kernels that together account for over 90% of GPU time in LLM inference. The generated kernels achieve 99–104% of the effective throughput of state-of-the-art hand-optimized assembly implementations and are 2–1543× faster than existing agentic systems in geometric-mean throughput across the evaluated workload families. Argus further generalizes to 200 KernelBench tasks, producing correct kernels for 100% of Level 1 and 90% of Level 2 problems.

1 Introduction

Efficient GPU kernel implementation is critical to large-scale LLM deployment. Given the hundreds of billions of dollars invested in GPU infrastructure, even single-digit improvements in kernel efficiency can translate into savings on the order of billions of dollars [61]. A well-optimized matrix multiplication (GEMM) on an NVIDIA A100 GPU can be 50× faster than a naïve implementation [8], but achieving this speedup requires coordinated optimizations across the full software stack: hardware matrix cores [41], software pipelining [34], memory hierarchy exploitation [37], and instruction scheduling [31]. Tensor compilers [15, 26, 42, 59, 63, 64, 72] automate subsets of this space through heuristic and evolutionary search, but for peak performance, hardware vendors ship hand-optimized assembly libraries [1, 4] that require months of engineering and struggle to keep pace with yearly hardware refreshes and evolving workloads [57].

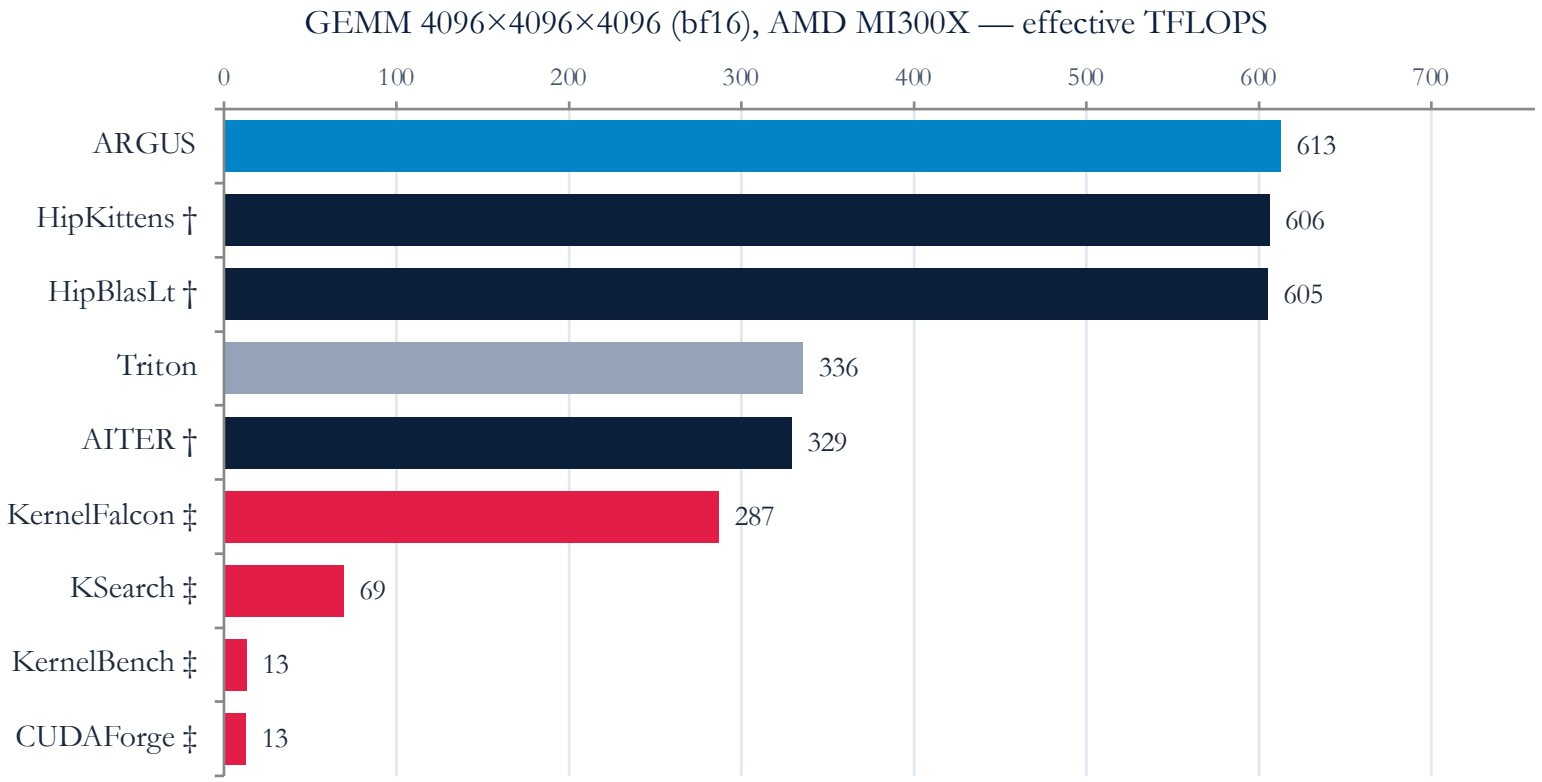
Recent work applies LLMs and coding agents to GPU kernel generation [6, 10, 17, 22, 23, 29, 38, 57, 62, 69], producing functionally correct kernels across a range of workloads [49]. However, the generated kernels are far from competitive compared with hand-optimized code on performance-critical workloads: on our AMD MI300X platform, the best agent-generated CUDA/HIP GEMM kernels are 2–600× slower than the state-of-the-art library. The gap is fundamental: correctness can be verified with tests, but high performance requires coordinated reasoning across software pipelining, instruction scheduling, register allocation, and NUMA-aware memory access, while accounting for the effects of memory aliasing [53] and synchronization [7]. Training data for such low-level optimizations is scarce, and the full kernel optimization trajectory often exceeds LLM context windows, where performance degrades [39]. Targeting Triton [59] narrows the gap to approximately 2× by offloading optimizations like software pipelining to the compiler, but the remaining 2×, including fine-grained instruction scheduling and warp specialization, lies below Triton's abstraction level.

In this paper, we present Argus, an agentic framework that automates complex, multi-level GPU kernel optimizations. Argus makes these optimizations robust through *data-flow invariants*: compile-time assertions that specify how

[arXiv: 2604.18616 (to appear in **SOSP 2026**)]

Correct Is Not Fast

> ***The gap.*** ***LLM agents already generate functionally correct kernels. On the workloads that dominate inference, they remain one to three orders of magnitude below hand-tuned libraries.***



† *hand-optimised library* ‡ *agentic kernel-generation framework*

2–600×

Best agent-generated GEMM vs. the state-of-the-art vendor library on MI300X.

> 90%

Share of LLM-inference GPU time spent in GEMM, attention, and Mixture-of-Experts.

≈ 2×

Residual gap even when an agent targets Triton — the rest lies below its abstraction.

Data-Flow Invariants in a Learnable DSL

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

> ***Idea.*** ***Data-flow invariants are compile-time specifications of how data must be choreographed throughout a kernel's execution — a technique adapted from information-flow security.***

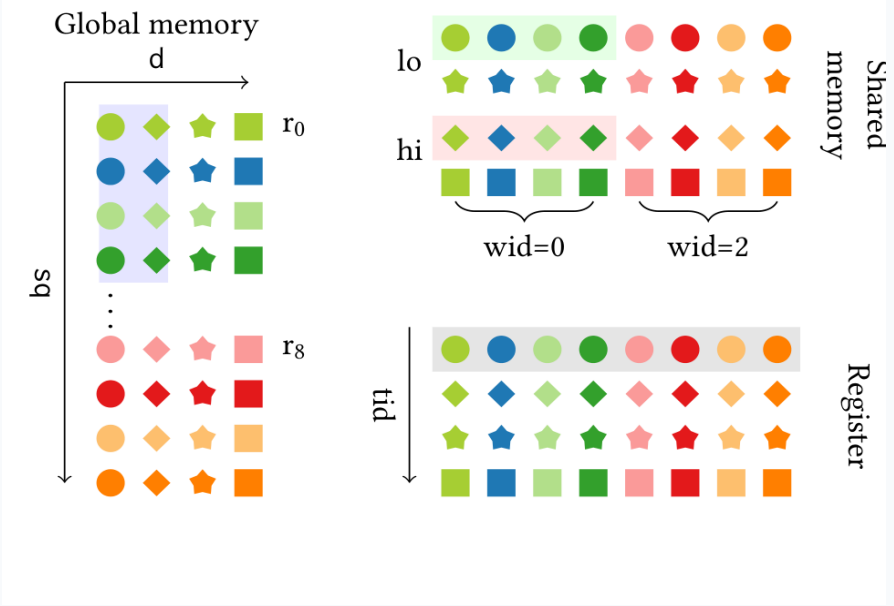
① Tag functions — attach symbolic tags (logical coordinates) to tensor elements; tags propagate through data and control flow.

```
# Assign tags to QKV
Γ_Q,Γ_K = Q[sq,h_q,d]⟶(sq%32,h_q/gqa,d), K[sq,h_kv,d]⟶(sq%32,h_kv,d)
Γ_V = V[sq,h_kv,d]⟶(sq,h_kv,d%32)
```

② Tag assertions — conformity: paired operands must match. Non-conformity: concurrent producers must not collide.

```
for j in range(16):
  tK,tQ = sK[j/8,wtid%32,j%8,wtid/32], rQ[j%8]
  tK,tQ = tK.view((8,), bf16), tQ.view((8,), bf16)
  assert tag(tK[...]) == tag(tQ[...]) # tQ and tK should match
```

The DSL. *Tile-based and Pythonic, syntactically close to CuTe, TileLang, and Triton, so the model can generalise from its training data. It exposes hardware instructions and compiler policy while hiding mechanism and intermediate representations — expressive enough for peak performance, concise enough to stay learnable.*



Tags for V propagating across memory levels: global ⟶ shared ⟶ register. Each colour–shape pair is one tag; the assertion at the MFMA use site checks the pairing survived.

[ARGUS, Figure 1]

The Agentic Optimisation Harness

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

Whole > parts. ***Tag functions specify; static analysis verifies; counterexamples reward. Alone, none suffices — together they close the loop.***

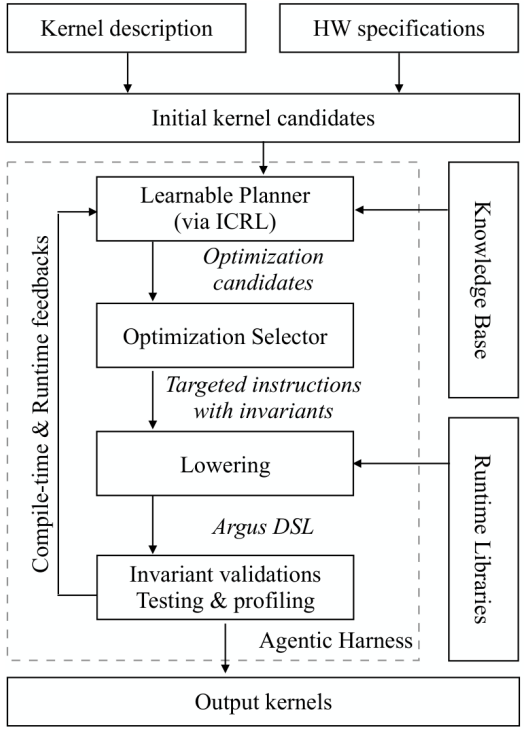
1. **Persistent knowledge base** Expert-curated. Each entry records a DSL transformation pattern and the invariants that must hold after the rewrite.

2. **Learnable planner (ICRL)** Binds abstract optimisations to the concrete kernel; emits ⟨optimization, context, score⟩ and specialises invariant templates into concrete tags and assertions.

3. **Optimization selector** Samples from the proposal distribution rather than taking the top rank — preserving exploration in a coupled space.

4. **Lowering agent** Implements the plan in the DSL at the level of tiles, layouts, and thread control; inserts the required tag assertions.

5. **Validator agent** Invariants → unit tests → runtime profile. Only candidates passing both static and dynamic checks are profiled.



[ARGUS, Figure 1 — agentic harness]

ICRL. ***The planner prompt is the mutable policy θ, updated by text gradients. Reward = runtime performance + process rewards from invariant violations. The knowledge base stays fixed and auditable; θ learns to bind it.***

ARGUS: Results

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

99–104%

of hand-tuned assembly throughput on GEMM, flash attention, and MoE.

2–1543×

geometric-mean gain over agentic baselines across workload families.

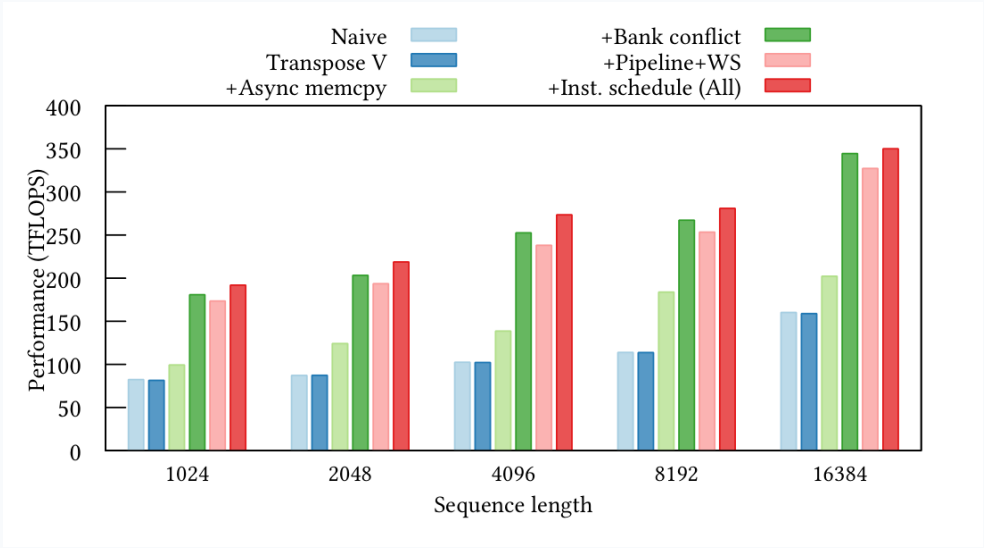
2.4×

over KernelFalcon, the strongest agentic baseline, on flash attention.

100 / 90%

of KernelBench Level-1 / Level-2 tasks correct — it generalises.

Ablation: cumulative optimisations on flash attention



[ARGUS, Figure 2]

Do the invariants earn their keep?

60 KernelBench GEMM / convolution problems; the same planner with and without invariants.

	Pass@1	Avg. tokens
Level 1	**60% → 75%**	18.1M → 15.0M
Level 2	**40% → 57%**	14.2M → 13.5M

More effective and cheaper at once: invariants raise the success rate while cutting token spend.

Argus is an agentic framework to implement every optimization class required to match expert-tuned production kernels.

Takeaway. ***Structured compile-time feedback is what closes the distance between an LLM agent and a human expert.***

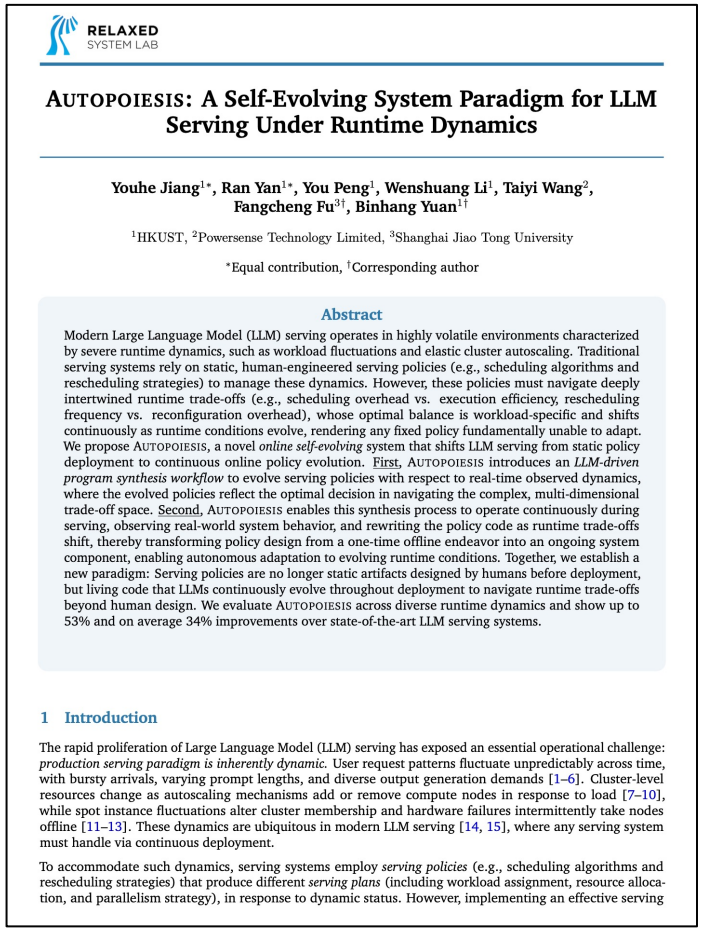
2 Autopoiesis

Online inference adaptation

Serving policies that an agent rewrites while the system is live.

Two-plane architecture · LLM-driven program synthesis · hot-swap

Homogeneous · heterogeneous · elastic cloud clusters



RELAXED SYSTEM LAB

AUTOPOIESIS: A Self-Evolving System Paradigm for LLM Serving Under Runtime Dynamics

Youhe Jiang[1]*, Ran Yan[1]*, You Peng[1], Wenshuang Li[1], Taiyi Wang[2], Fangcheng Fu[3]†, Binhang Yuan[1]†

[1]HKUST, [2]Powersense Technology Limited, [3]Shanghai Jiao Tong University

*Equal contribution, †Corresponding author

Abstract

Modern Large Language Model (LLM) serving operates in highly volatile environments characterized by severe runtime dynamics, such as workload fluctuations and elastic cluster autoscaling. Traditional serving systems rely on static, human-engineered serving policies (e.g., scheduling algorithms and rescheduling strategies) to manage these dynamics. However, these policies must navigate deeply intertwined runtime trade-offs (e.g., scheduling overhead vs. execution efficiency, rescheduling frequency vs. reconfiguration overhead), whose optimal balance is workload-specific and shifts continuously as runtime conditions evolve, rendering any fixed policy fundamentally unable to adapt. We propose AUTOPOIESIS, a novel *online self-evolving* system that shifts LLM serving from static policy deployment to continuous online policy evolution. First, AUTOPOIESIS introduces an *LLM-driven program synthesis workflow* to evolve serving policies with respect to real-time observed dynamics, where the evolved policies reflect the optimal decision in navigating the complex, multi-dimensional trade-off space. Second, AUTOPOIESIS enables this synthesis process to operate continuously during serving, observing real-world system behavior, and rewriting the policy code as runtime trade-offs shift, thereby transforming policy design from a one-time offline endeavor into an ongoing system component, enabling autonomous adaptation to evolving runtime conditions. Together, we establish a new paradigm: Serving policies are no longer static artifacts designed by humans before deployment, but living code that LLMs continuously evolve throughout deployment to navigate runtime trade-offs beyond human design. We evaluate AUTOPOIESIS across diverse runtime dynamics and show up to 53% and on average 34% improvements over state-of-the-art LLM serving systems.

1 Introduction

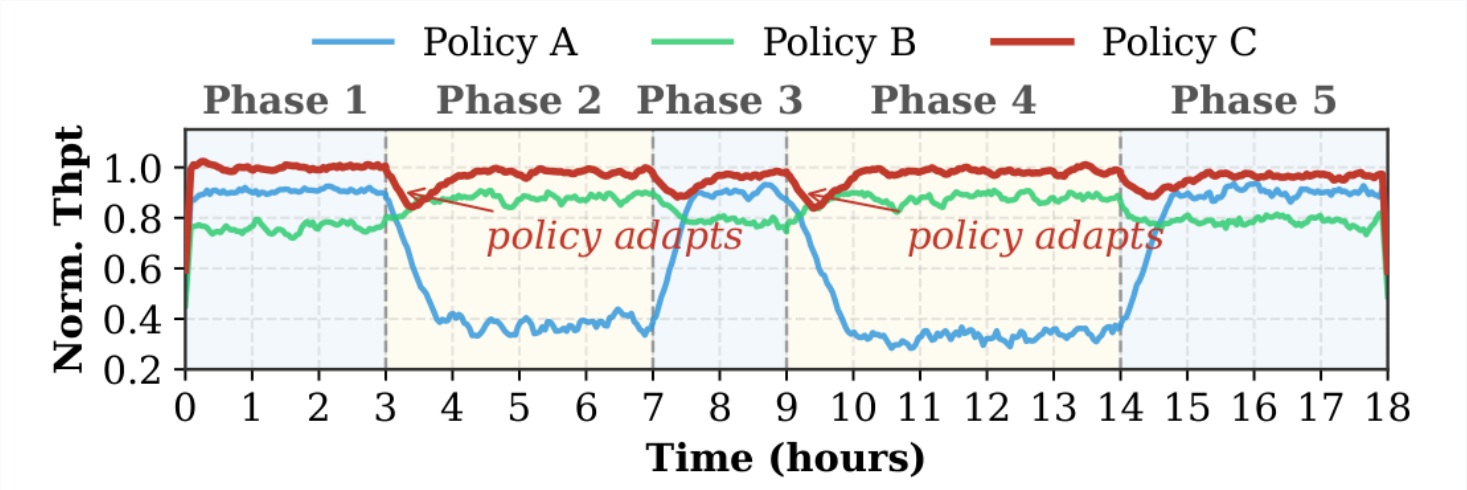
The rapid proliferation of Large Language Model (LLM) serving has exposed an essential operational challenge: *production serving paradigm is inherently dynamic.* User request patterns fluctuate unpredictably across time, with bursty arrivals, varying prompt lengths, and diverse output generation demands [1–6]. Cluster-level resources change as autoscaling mechanisms add or remove compute nodes in response to load [7–10], while spot instance fluctuations alter cluster membership and hardware failures intermittently take nodes offline [11–13]. These dynamics are ubiquitous in modern LLM serving [14, 15], where any serving system must handle via continuous deployment.

To accommodate such dynamics, serving systems employ *serving policies* (e.g., scheduling algorithms and rescheduling strategies) that produce different *serving plans* (including workload assignment, resource allocation, and parallelism strategy), in response to dynamic status. However, implementing an effective serving

[ArXiv: 2604.07144]

When Static Policies Break

The problem. Serving systems react to runtime dynamics by swapping plans — but the policy that produces those plans is static and human-engineered, so it cannot follow the trade-offs as they shift.



Five phases: steady (1, 3, 5) alternating with bursty (2, 4). [Autopoiesis, Figure 1]

Policy A
Thorough scheduling, aggressive reconfiguration. Excels under stability; collapses under bursts.

Policy B
Reactive scheduling, lightweight reconfiguration. Thrives during bursts; underperforms in steady state.

Policy C
Continuously adapts to shifting conditions. Holds high throughput across every phase.

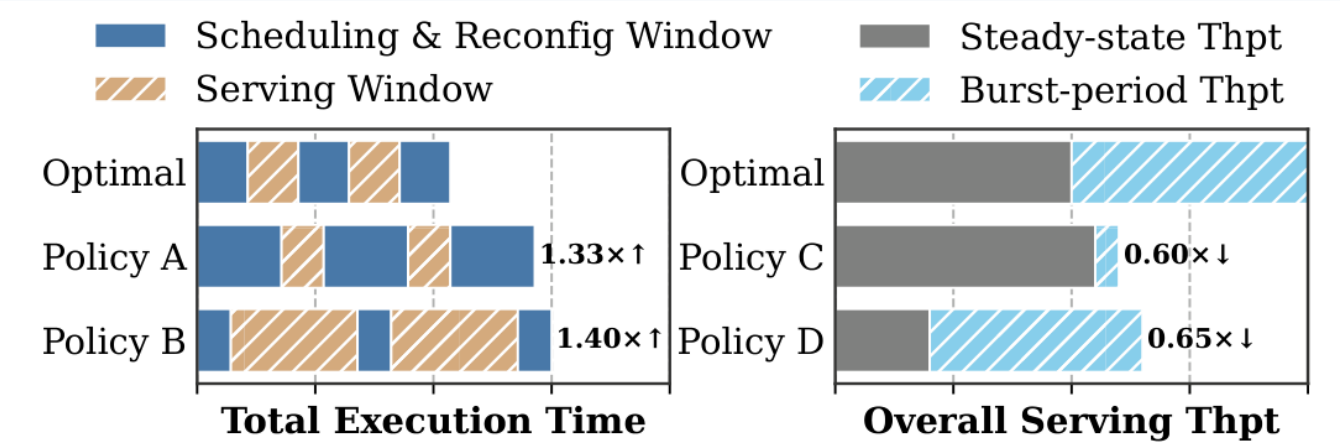
No fixed policy wins across regimes — the optimum is workload-specific and moves.

- **Human intuition cannot locate balances in a space this intertwined.**
- Offline precomputation cannot rescue it either — the space is not enumerable in advance.

An Intertwined Trade-Off Space

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
RELAXED SYSTEM LAB

> ***Root cause.*** ***Serving never pauses. Under this continuous-execution constraint every scheduling decision and every reconfiguration action has a direct, concurrent cost — which couples the trade-offs together.***



Left: neither extreme is competitive (1.33× / 1.40× worse than the oracle). Right: a policy tuned for one regime loses 35–40% in the other. [Figure 2]

Two requirements for an effective serving paradigm

- Automated trade-off exploration — discover policies tailored to the current workload and cluster, rather than balancing the axes by intuition.
- Continuous online evolution — observe real system behaviour and rewrite the policy code as the trade-offs shift, throughout deployment.

Rescheduling frequency vs. per-interval overhead

Frequent rescheduling (large N) keeps plans fresh but accumulates cost; infrequent rescheduling serves under stale plans.

Scheduling thoroughness vs. stale serving cost

Thorough scheduling lowers t_serve but extends the stale-plan window t_stale; lightweight scheduling inverts the pair.

Reconfiguration aggressiveness vs. transition overhead

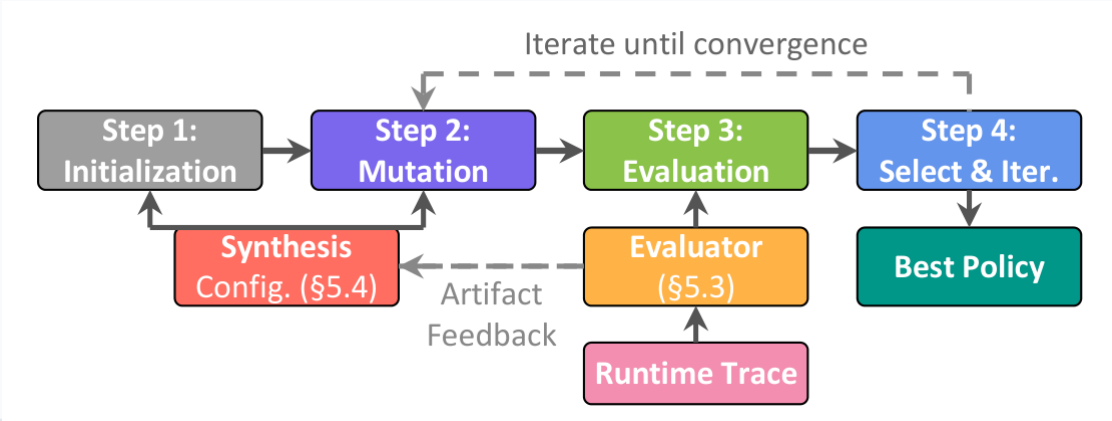
Aggressive reconfiguration reaches the optimal plan but pays prolonged degradation proportional to the change.

Trade-offs (i)–(iii) are tightly coupled: the optimum is nonlinear, deployment-specific, and moves continuously.

LLM-Driven Program Synthesis

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
RELAXED SYSTEM LAB

> ***The reward signal.*** ***What compile-time counterexamples are to ARGUS, a calibrated simulator and a structured cost breakdown are to Autopoiesis — the same dense-feedback move, one layer up.***



Artifact feedback: the cost breakdown

	N	Σ stale	Σ reconf	Σ serve	T_total
Parent	2	7.6 s	12.0 s	25.2 s	**44.8 s**
Child A	5	5.5 s	4.0 s	24.0 s	**33.5 s**
Child B	1	10.0 s	17.0 s	23.0 s	**50.0 s**

The agent sees which axis its own edit moved: Child A reschedules more often yet pays less — lightweight, frequent rescheduling suits this workload.

The evaluator is the bridge to reality

- Candidates are replayed against the actual runtime trace snapshotted from the data plane — not a synthetic benchmark.
- A roofline-based simulator models per-operator time, continuous batching, parallelism, and reconfiguration overhead.
- Selection uses MAP-Elites with island-based population management, preserving diversity while retaining elites.

Trade-off-aware mutation

- Formal structure — the system prompt encodes the execution model and the trade-off characterisations, so the LLM knows how a move on one axis shifts the others.
- Per-iteration feedback — the parent/child cost breakdown plus population context, steering away from redundant mutations.
- **Result: trade-off-aware modifications, not random variations.**

Making Evolution Truly Online

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
RELAXED SYSTEM LAB

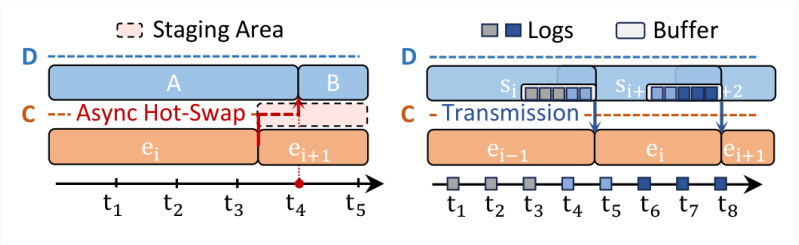
Engineering. ***Offline synthesis is a research artefact; online synthesis is a systems problem. Four mechanisms make the control plane fast enough and safe enough to run beside live traffic.***

1. **Warm-start re-evolution** Consecutive snapshots reflect gradual shifts, so cycle e·i is seeded with the elites of e·i−1 and their mutations — targeted refinement instead of re-exploring from scratch.

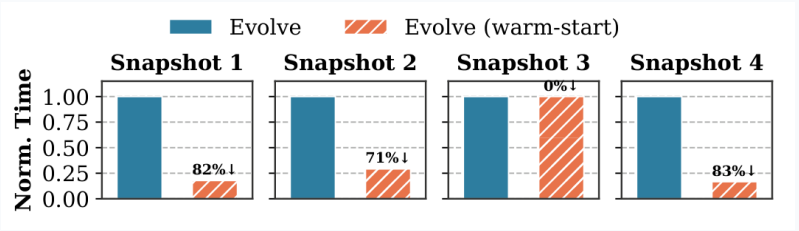
2. **Multi-level timeout hierarchy** An evolution-level timeout bounds each cycle so the control plane delivers on a predictable latency; a candidate-level timeout kills any schedule function that diverges.

3. **Asynchronous policy hot-swap** Because the policy is two pure functions, deployment is a code replacement. The control plane writes to a staging area; the data plane loads it at the next monitoring step. No in-flight plan is disrupted.

4. **Sliding-window snapshotting** Conditions are logged to a fixed-size circular buffer; a background thread transmits a contiguous window on request, decoupled from live request processing. Windows may overlap, exposing trends across cycles.



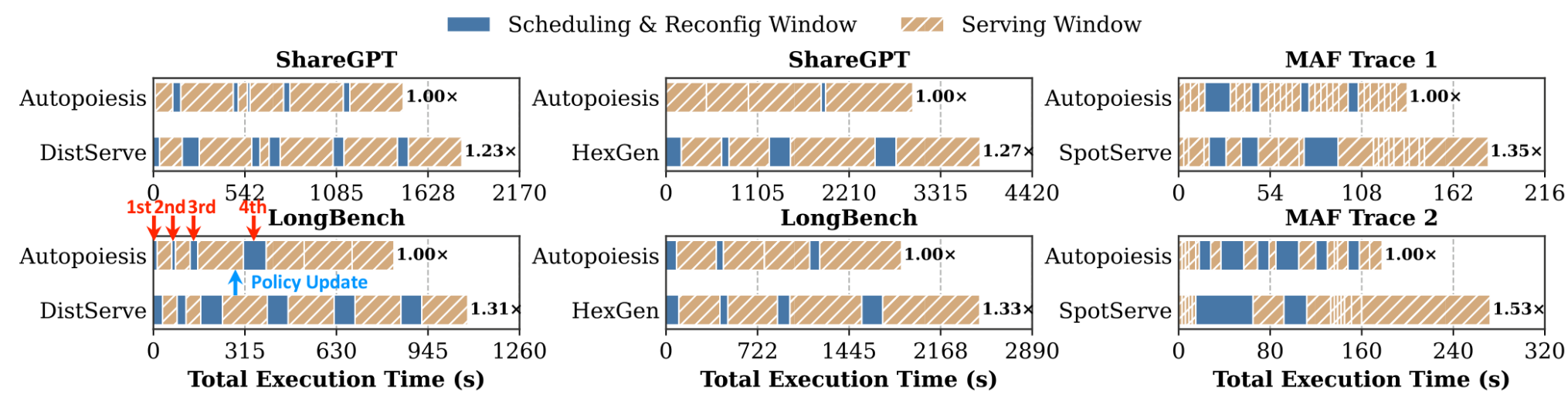
Hot-swap (left) and trace snapshotting (right). [Autopoiesis, Figure 6]



Warm-start cuts evolution time by up to 83% — except when conditions shift so far that no prior structure is reusable (Snapshot 3). [Figure 9]

Convergence in under 10 minutes (≈ 80 iterations); evaluation is embarrassingly parallel — up to 8× with 10 threads.

Autopoiesis: Results



End-to-end execution time vs. DistServe (homogeneous), HexGen (heterogeneous), SpotServe (elastic cloud). [Autopoiesis, Figure 7]

53% / 34%

Peak / average reduction in execution time over state-of-the-art systems.

27%

Average gain over DistServe on a homogeneous cluster (up to 31%).

30%

Average gain over HexGen under hardware heterogeneity (up to 33%).

44%

Average gain over SpotServe on an elastic cloud cluster (up to 53%).

The evolved policies are workload-adaptive

- Under volatility: cuts scheduling and reconfiguration overhead by 65%.
- Under stability: invests 10% more in search — cutting serving latency 22%.
- On heterogeneous clusters, search time drops by up to 92%.

Two insights from the case studies

- Frontier-shifting, not frontier-navigating. Greedy and ILP occupy fixed points on a Pareto frontier; evolved policies improve both axes at once.
- Scheduling algorithm and rescheduling strategy must co-evolve — their cost profiles are coupled.

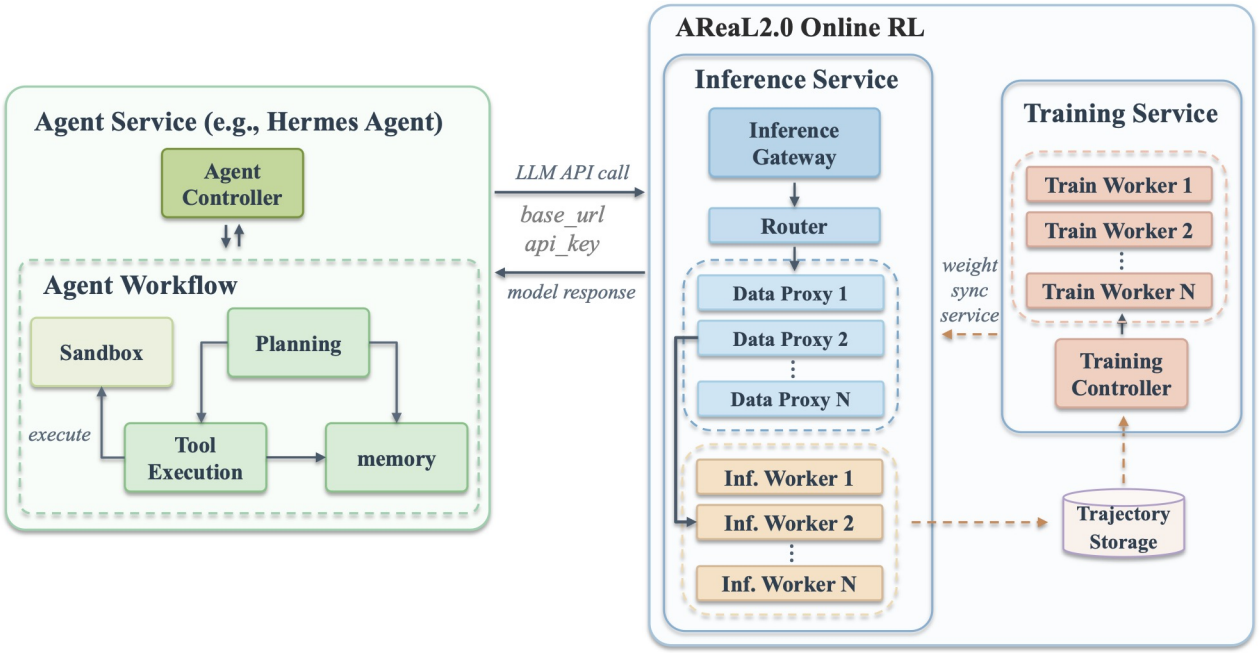
3 AReaL-AutoPilot

Full lifecycle RL tasks (in progress)

An agentic control layer for distributed RL post-training.

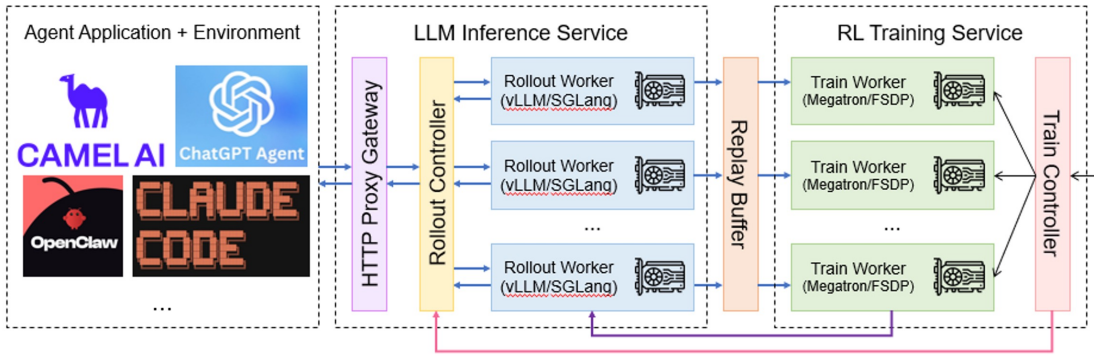
Spec → validated plan → launch → monitor → adapt

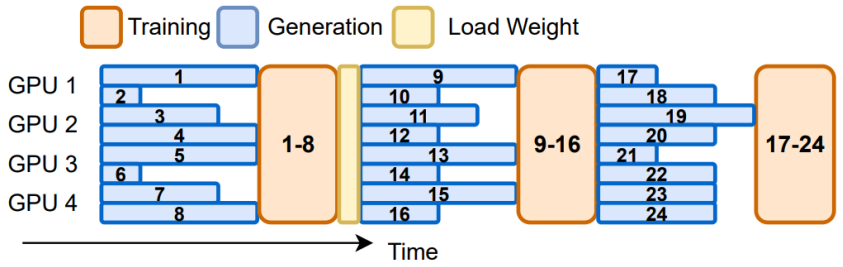
Built on AReaL; automatic RL deployment.

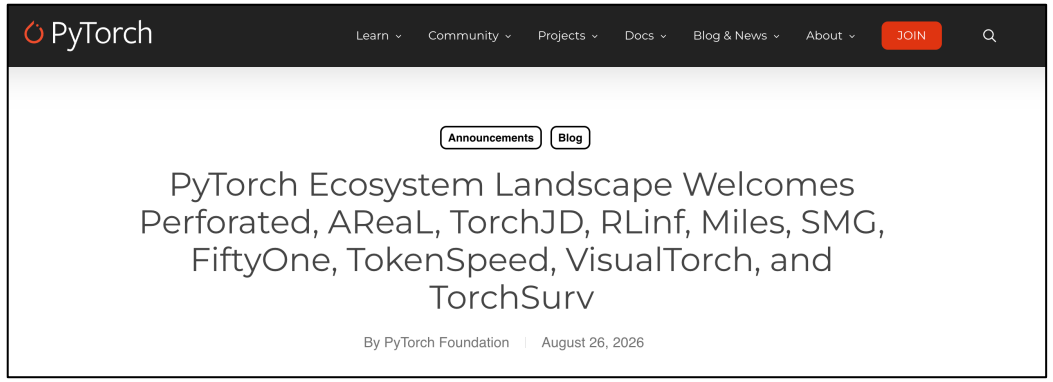


AReaL 2.0 online-RL architecture — the execution substrate that AutoPilot configures, launches, and adapts.

AReaL: Open Source Community







What is AReaL?

A scalable agentic reinforcement-learning (RL) infrastructure bridging foundation-model training with modern agent-based applications.

Jointly open-sourced by Ant Group and Tsinghua University · February 2025

- **Flexibility** — modular architecture with plug-and-play "Agentic RL-as-a-Service" integration.
- **Scalability** — stable, fully asynchronous RL training at large scale.
- **Performance** — state-of-the-art results on math, code, search, and agentic tasks.
- **Research-friendly** — four major RL paradigms and 10+ mainstream RL algorithms.
- **PyTorch Ecosystem project** — accepted May 2026; now applying for Foundation hosting.

AReaL: Community & Governance

5.7K+	500+	100+	1,900+	4 → 14
GitHub stars	Forks	Code contributors	Community users	Maintainers since launch

Vendor-Neutral Governance

- **Maintainer-led model,** publicly documented in GOVERNANCE.md.
- **14 maintainers across 6 organizations** — Tsinghua, Ant Group, Huawei, ByteDance, Xiaomi, and Tencent.
- **Two maintainer approvals per pull request,** including a code owner for modified paths.
- **Consensus decision-making,** with a lead-maintainer tie-break when consensus cannot be reached.
- **Gold OpenSSF Best Practices badge** (bestpractices.dev/projects/12770).
- **Project assets prepared** for transfer to the Linux Foundation.

Contributor & Maintainer Organizations

Maintainers: Tsinghua · Ant Group · Huawei · ByteDance · Tencent · Xiaomi
Contributors incl. Microsoft, Boson AI, PKU, CUHK, HKUST, UIUC

AReaL: Release History & Cadence

AReaL | THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY | RELAXED SYSTEM LAB

Every release ships three artifacts:

1. **Versioned code release**
   Tagged, installable, and reproducible.
2. **Reproducible training examples**
   End-to-end recipes with configs and scripts.
3. **Research findings & papers**
   Empirical results published with each milestone.

Online RL

- **August 2026** — **PyTorch Ecosystem project** — announced
- **Jun 2026** — **v2.0** Online-RL modular architecture

Agentic RL

- **Mar 2026** — **v1.0** Agentic architecture — general availability
- **Dec 2025** — **v0.5** Agentic architecture preview

Reasoning RL

- **Aug 2025** — **v0.4** AReaL-lite architecture
- **Jun 2025** — **v0.3** Fully asynchronous RL — 2.7× throughput
- **Mar 2025** — **v0.2** State-of-the-art 7B math model
- **Feb 2025** — **v0.1** Initial release with GRPO

18 months from v0.1 to v2.0 — three architecture generations: Reasoning → Agentic → Online RL.

AReaL-AutoPilot: The Full Lifecycle

> ***Goal.*** ***An agentic control layer that turns an RL workload specification into a validated deployment plan, launches the distributed system, monitors it, and progressively adapts its policies.***

1 Workload spec

Model, algorithm, dataset, resource envelope, objective.

2 Validated plan

Parallelism, placement, batching, rollout/training ratio — checked before launch.

3 Configure & launch

Instantiate the distributed RL system on the cluster.

4 Monitor

Throughput, staleness, memory, stragglers, convergence.

5 Adapt

Rewrite system policies as models, workloads, and hardware change.

progressive adaptation — the plan is never final, only current

Inherits from ARGUS · validate before you act

- A deployment plan is checked against system invariants — memory feasibility, parallelism legality, communication topology — before a single GPU is allocated.
- A rejected plan returns a counterexample, not a crash log: the agent learns which constraint it violated.

Inherits from Autopoiesis · adapt while you run

- RL post-training is itself non-stationary: rollout length drifts as the policy improves, so the optimal system configuration drifts with it.
- The control plane observes execution and rewrites system policies mid-run — the same two-plane separation, applied to training.

Conclusion

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

RELAXED SYSTEM LAB

> *The arc. From code-generation assistant to autonomous system engineer — offline (ARGUS), online (Autopoiesis), and across the full lifecycle (AReaL-AutoPilot).*

ARGUS

Data-flow invariants turn a compiler into a teaching signal — reaching 99–104% of hand-tuned assembly on GEMM, attention, and MoE.

Autopoiesis

Serving policies become living code, evolved online — 34% average and up to 53% improvement over state-of-the-art systems.

AReaL-AutoPilot

Both halves, applied to distributed RL post-training: validate before launch, adapt while running.

AReaL Repo:
https://github.com/areal-project/AReaL

Personal page:
https://binhangyuan.github.io/site/

Thank you!